Instituto Universitário de Lisboa

Department of Information Science and Technology

Generalized software application for operation of a 3D vehicle in air, water and land

Diogo Rafael Baptista Peres

A dissertation presented in partial fulfillment of the Requirements for the Degree of

Master in Computer Science Engineering

Supervisor:
PhD. Pedro Joaquim Amaro Sebastião,
Assistant professor at ISCTE-IUL

Co-Supervisor:
PhD. Nuno Manuel Branco Souto,
Assistant professor at ISCTE-IUL

June, 2017

ABSTRACT

The unmanned vehicles (UV) and its applications are growing exponentially. Using the radio control is the most common way to control these types of vehicles for being a simple and cheap method to control an UV. However, it doesn't have a visual interface that allows the user to see the vehicle's information such as battery status, speed, distance, geolocation, etc. To deal with this problem, some mobile and desktop applications have been developed. To communicate between the control device and the vehicle, dongles are commonly used to establish the connection using radio, Bluetooth or Wi-Fi. In most cases, these technologies don't allow the user to control at long distances, Beyond Line-Of-Sight (BLOS), and these applications are focused to use mostly on multi-copters, and most of the times, they only allow to connect a vehicle at a time.

The purpose of this dissertation is to study the reliability of an application able to control multiple types of vehicles, such as aerial, land and water vehicles. This application allows the user to connect multiple vehicles at the same time using a single device, easily change the vehicle assigned to control, by using mobile networks to perform the communication between the developed application and the vehicle. In this way, it will be possible to connect a 3D – hybrid vehicle, which is a vehicle capable of moving in water, land and air environments, allowing the user to control the vehicle at long distances with video feedback.

To achieve the purpose of this dissertation, it was developed an Android application to allow controlling the vehicle by using mobile networks to communicate. In the vehicle, besides the common electronics used in an unmanned vehicle (ESC's, motors, batteries, controller board, etc.), it will be used a Raspberry Pi 2 model B with a 3rd Generation (3G) and 4th Generation (4G) dongle that will connect the vehicle to the internet, routing the messages coming from the controller board placed in the vehicle to the mobile application. It was also developed a server application to do the user management and exchange the messages coming from both platforms: vehicle and application.

Key-words: vehicle 3D, Drone, unmanned vehicles, Android application, multi-vehicle, mobile networks.

This page was intentionally left in blank

RESUMO

Os veículos não tripulados e as suas aplicações estão em forte crescimento. O uso de rádio controlo é a maneira mais comum de controlar estes tipos de veículos, sendo o método mais barato e simples de controlar um veículo não tripulado. Contudo, não têm uma interface visual que permita ao utilizador ver as informações do veículo, tais como o nível da bateria, a velocidade, distância, geolocalização, entre outros. Para ajudar com este problema, têm sido desenvolvidas algumas aplicações para dispositivos móveis e computadores, que permitem controlar e monitorizar este tipo de veículos. Para estabelecer a comunicação entre o dispositivo de controlo e o veículo, são frequentemente usados *dongles* para comunicar por rádio, *Bluetooth* ou *Wi-Fi*. Na maioria dos casos, estas tecnologias não possibilitam ao utilizador o controlo a longas distâncias, para além da linha de vista, e costumam ser focadas para o uso em multicopteros, possibilitando, na maioria dos casos, a ligação de um único veículo.

O âmbito desta dissertação pretende estudar e desenvolver uma aplicação com elevada fiabilidade, capaz de controlar vários tipos de veículos, nomeadamente, veículos aéreos, terrestres e aquáticos. Esta aplicação irá permitir a ligação a vários veículos ao mesmo tempo, trocar facilmente o veiculo a controlar, recorrendo aos sistemas de comunicação móveis celulares, 3ª geração (3G) e 4ª geração (4G) para garantir a comunicação entre a aplicação desenvolvida e o veículo não tripulado. Seguindo estes princípios, é possível controlar um veículo 3D hibrido (em modo de ar, terra e mar). Esta permite ao utilizador controlar o veículo a longas distâncias com o uso de uma transmissão de vídeo. Para alcançar o objetivo desta dissertação foi desenvolvida uma aplicação Android para possibilitar o controlo recorrendo às redes móveis celulares. No veículo, além da eletrónica habitual, para um veículo não tripulado (motores, ESC's, baterias, etc.), será também utilizado um Raspberry Pi 2 modelo B com um *dongle* 3G/4G que liga o veículo, redirecionando as mensagens vindas da placa de controlo para a aplicação móvel. Para a comunicação entre a aplicação e o veículo foi também desenvolvida uma aplicação instalada no servidor que é responsável pela gestão de utilizadores e pela troca de mensagens vindas de ambas as plataformas: veículo e aplicação.

Palavras-chave: veiculo 3D, Drone, Aplicação Android, veículos não tripulados, multi-veículo, redes móveis.

This page was intentionally left in blank

ACKNOWLEDGMENTS

I would like to thank my mother for her support and motivation and my father for being a great inspiration to achieve greater goals.

I would like to thank my supervisor, Professor Pedro Sebastião, for all the support and knowledge, in my work and personal life, always available to help and encouraging to do more and better. For finding more ways to enrich my curriculum, by giving workshops, or going to events to present the projects that were being developed in ISCTE-IUL and IT.

To my co-supervisor Professor Nuno Souto for his technical knowledge and advices.

To my good friend António Raimundo, for helping with his knowledge throughout the development of this dissertation. For helping me finding solutions to some of the problems encountered.

To Nuno Santos for helping with his knowledge and accompaniment in the testing phase.

Also, I would also like to thank my friend Manuel Oliveira for his advices to optimize the server and also for his good mood.

And last, but not least, my friend Ricardo Silva for his knowledge in drones and electronics components and also by his help to mount and test the drones in a better and safer way.

This page was intentionally left in blank

ACRONYMS & ABBREVIATIONS

3G	3rd Generation
4G	4th Generation
ACK	Acknowledgement
API	Application Programming Interface
APM	Ardupilot Mega
BEC	Battery Eliminator Circuits
BLOS	Beyond Line-Of-Sight
CPU	Central Processing Unit
DC	Direct Current
ESC	Electronic Speed Controller
GCS	Ground Control Station
GPIO	General Purpose Input Output
GPS	Global Positioning System
ID	Identifier
LED	Light Emitting Diode
LGPL	GNU Lesser General Public License
LiPo	Lithium Polymer
MAV	Micro Aerial Vehicle
MAVLink	Micro Air Vehicle Link
NAT	Network Address Translation
NiCd	Nickel Cadmium
NiMH	Nickel-Metal Hydride
RAM	Random Access Memory
RC	Radio Control
RPi	Raspberry Pi
RSSI	Received Signal Strength Indicator
SITL	Software In The Loop
TCP	Transmission Control Protocol
TV	Television
UAV	Unmanned Aerial Vehicle
UBEC	Universal Battery Eliminator Circuit

UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicle
USB	Universal Serial Bus
USV	Unmanned Surface Vehicle
UV	Unmanned Vehicle
V4L	Video for Linux
VPN	Virtual Private Network

CONTENTS

This page was intentionally left in blank

FIGURES

TABLES

This page was intentionally left in blank

Chapter 1
INTRODUCTION

This chapter will introduce the scope of this dissertation by describing a general overview, its structure and goals, as well as explaining the main motivation for its realization.

1.1 Overview

An unmanned vehicle (UV) is a vehicle without a person on board. This type of vehicles can be remotely controlled, remotely guided (by sending coordinates) or autonomous vehicles which are capable of sensing their environment, navigate and make decisions on their own. This dissertation will approach three main types of UVs, Unmanned Ground Vehicle (UGV), Unmanned Aerial Vehicle (UAV) and Unmanned Surface Vehicle (USV), with the goal of simulating a 3D vehicle, capable of moving in three environments.

In this dissertation, it will be used an on-board controller, capable of controlling all the three types of UV's mentioned above. This board is composed by several sensors, such as a gyroscope, an accelerometer, a barometer, a magnetometer and a Global Positioning System (GPS). The on-board controller receives the commands from the transmitter and interprets it to control the motors, while maintaining the vehicle stable. With the assistance of these sensors, the controller board is also able to drive the vehicle through waypoints by itself.

To control the vehicle at a distance, it's generally used a radio controller that is the simplest and easiest way of doing it. However, the radio controller has a short range of communication and cannot show some information such as the geolocation of the drone or its velocity. To overcome this problem, the control can be established by satellite, but due to its expensive price, it's only used to military purposes. It has been developed a few solutions to overcome this problem, using wireless networks like Wi-Fi or 3G/4G. However, it have been focused in just one type of UV. This dissertation will focus on developing a mobile application software in order to be able to control an UV capable of moving in ground, aerial and surface environments through wireless communications. This requires dealing with the differences among the types of vehicles and their driving type.

1.2 Motivation

In recent years, the use of unmanned vehicles has grown quickly. The technology was developed primarily to use in the military area. The ability to use them in dangerous missions without risking human lives was the main factor that led to its creation. The success of this technology aroused the curiosity of people who quickly realized the potential of these technologies. Nowadays, unmanned vehicles can do all kinds of tasks. In the military field, they can participate in rescue operations, surveillance, explosives disarming, pursuit operations, delivering supplies to remote and/or inaccessible regions, border patrol missions, crowd monitoring, and so on. They have also a wide range of applications for civilian and commercial use, such as forest fire detection, counting wildlife, analysis and treatment of crops, aerial imaging/mapping, environmental monitoring, filmmaking, photo capture, racing sports, stunts and much more (Nehme, 2009). Its great utilities have led to a fast growth of the UV market.

Unmanned Aerial Vehicles sector is one of the most growing sectors of the world aerospace industry in this decade. In 2015 its production was expected to soar from $4 billion to $14 billion annually, totaling $93 Billion in the next ten years, and military UAV research would add another $30 Billion over this period. Philip Finnegan, director of Corporate Analysis at Teal Group, said that "the civil UAV market continues to grow with each annual report, mirroring the increase in the civil market itself" (UAV Production Will Total $93 Billion, 2015). Precision agriculture and public safety are the most promising commercial and civil markets for Unmanned Aircraft Systems (UAS), owning 90% of its known potential market. The economic impact of the UAS in the national air space will reach $82.1 billion between 2015 and 2025. Figure 1-1 shows the total expected sales for 2015-2025 (Jenkins & Vasigh, 2013).

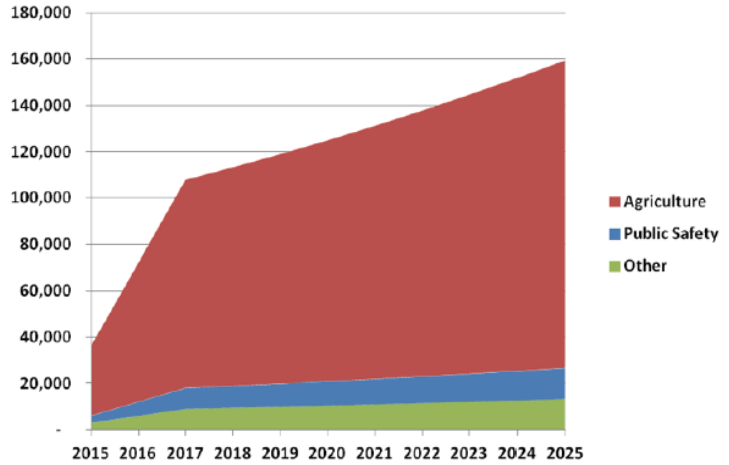


Figure 1-1 – Annual UAS sales for agriculture, public safety, and other markets.
Source: (Jenkins & Vasigh, 2013)

For Unmanned Ground Vehicles there's a similar scenario. The market for UGV is expected to growth at a compound annual growth rate of 23.7%, growing from $6.44 Billion in 2015 to $18.65 Billion by 2020 (Unmanned Ground Vehicles (UGV) Market by Application (Defense, Commercial), Mode of Operation, Size, Mobility, Payload, and Region (North America, Europe, Asia-Pacific, Middle East, Rest of the World) - Global Forecasts to 2020., 2016). As well as in the UAV and UGV markets, there's also been a growth in the global market for Unmanned Surface Vehicles ("Unmanned Surface Vehicles for Defense and Security", 2012). This market growth has led to an increase development of new technologies in the field of unmanned vehicles.

1.3 Goals

The main goal of this dissertation is to develop a system with a mobile application able to control a 3D vehicle, a more versatile vehicle, able to move in the three dimensions: land, water and air. To achieve this objective a series of objectives were established:

- Develop a server responsible for the client management, to register the users and their vehicles. The server will be responsible for dealing with the messages coming from both ends and redirect them to their pairs;
- Establish a connection to the vehicle via wireless networks;
- Create a mobile application to control the vehicle with the help of video feedback, making it user friendly and easy to use as well as keeping a good performance in Beyond Line-of-Sight (BLoS);

- Develop the application to control different types of vehicles, and also a 3D vehicle, operating in 3 modes. For this purpose, it will be necessary to develop or build different types of vehicles to test the application.

1.4 Research Questions & Methods

During the planning and the development phase of this dissertation, several questions were raised, which had to be answered correctly in order to achieve the proposed goals. The questions raised were the following:

- How to communicate between the application and the vehicle in an efficient way, reducing latency to its lowest value possible, even when there are multiple vehicles connected at the same time?
- How to allow the control of a 3D vehicle?
- What is the best communication protocol to communicate with the controller board in the vehicle?
- Which is the best method to transmit the video between the application and the vehicle, minimizing the time delay?
- How to develop a software able to deal with the different types of UVs, distinguishing and recognizing these vehicles when connected at the same time?

To develop this dissertation and answer the questions described above, it was used an agile and experimental methodology. During the vehicle's assembly and in the development phase of the application, multiple tests were done constantly in order to analyze and compare their performance with examples found in other dissertations and in the internet, making constant improvements to achieve a better performance when compared with other solutions.

1.5 Contributions

Throughout the development of this dissertation, with the knowledge acquired from its realization, it was possible to give several voluntary contributions, promoted by the supervisor of this dissertation, Professor Pedro Sebastião. In order to spread the work that was being developed by a R&D group, IT (Institute of Telecommunications) and to participate in the realization of several events. All these contributions were very self-gratifying since it helped people to understand better the world of unmanned vehicles, understanding its problems and challenges. It was also a good opportunity to meet new

people from the different areas in which this dissertation is inserted. The contributions given were:

- Photogrammetry of Batalha's Monastery facade, located in Batalha, in the district of Leiria, Portugal. This work was done at the invitation of an architecture student in collaboration with the Portuguese Cultural and Patrimony Department (DGPC), and the College of Architecture of the University of Lisbon (FAUL). The goal was to use a custom made UAV to take several photos of the monastery facade, in different angles, in order to create a virtual 3D model to analyze the possible problems with the facade.
- Presentation of the projects being developed by IT in the 'Encontro Ciência 2016', that took place in the Congress Center of Lisbon, from July 4th to 6th and aims to promote the work being developed in Portugal, in the scientific and technological areas.
- Tree planting in Montejunto forest, Portugal. This event held by ISCTE, called *ISCTE-IUL Carbon 0%*, was done with the objective of improving sustainability and taking a step towards reaching the carbon neutrality. During this day, several drone flights were done to record and publicize the event, which helped to understand better some of the challenges of flying a drone.
- Presentation of three workshops focused on the assembly of several customized drones. The drones were made from scratch, using different components. After their construction it was taught how to control and fly a drone.
    - The First workshop was done at ISCTE-IUL Academy Days, which occur from March 31th to April 1st of 2016 and consisted of multiple workshops for students of high schools, with the purpose of presenting the college and the work that is being done.
    - The second workshop was inserted in the "Electronic Laboratories Week" at *Universidade da Beira Interior* in June 30th, 2016.
    - The third and last workshop was inserted in the "Drone's Week", that took place at ISCTE-IUL, from June 18th to June 22nd, 2016, and it was supported by *Ciência Viva.* This event has the purpose to spread the knowledge about drones, with several lectures and workshops throughout a week.

- A paper was accepted to *ICUAS'17,* The *2017 International Conference on Unmanned Aircraft Systems.* This conference takes place from June 13th to June 16th and will be held in Miami, Florida, United States of America.
- A paper was accepted to *UAV-g 2017 – International Conference on Unmanned Aerial Vehicles for Geomatics*, that will be held in Bonn, Germany, from September 4th to September 7th of 2017;
- A paper was submitted to *RED-UAS 2017 – Research, Education and Development on Unmanned Aerial Systems* that will happen in Linköping, Sweden on October 3rd to 5th.

1.6 State of Art

As technology grows the hardware is faster and smaller. Currently the companies have developed a variety of software and hardware to simplify the control of UVs, besides that, there's a lot of open source projects developed to simplify and provide more ways to control UVs.

1.6.1 Unmanned Air Vehicles

An Unmanned Aerial Vehicle commonly known as a drone, is an aircraft without a human pilot aboard. There are a few different types of UAVs, all with different flight dynamics. This dissertation will be focused in the Micro Aerial Vehicle (MAV), such as the multi motor or multicopter, a mechanically simple aerial vehicle whose motion is controlled by speeding or slowing multiple downward thrusting motor/propeller units. This vehicle is capable of take-off vertically, very stable and has a good loading capacity, some are even capable of keep flying and landing safely with one malfunctioning motor. However, this type of vehicle requires constant adjustments in the motors to keep it stabilized and due to his number of motors, there is a high energy consumption. In the MAV category, there's also the helicopter, with just one motor, with less payload capability and less battery waste (Saraiva, 2015). At last, we have the fixed-wings, which can take-off horizontally and are able of gliding without losing much height, even without propulsion. This design makes it impossible to hover on a certain position. The only alternative is to circle around a given position. Fixed-wings don't have a lot of payload capacity, however, they consume less battery, ideal for long distance missions. Positional changes are achieved as with a regular airplane, e.g. thrust, rudder, elevators and ailerons (Scherer, 2015).

Figure 1-2 – Example a fixed wing, Bixler 2.
Source: (HobbyKing Bixler 2 EPO 1500mm w/ Motor, Servos and Optional Flaps (ARF), 2016)

Figure 1-3 – Example of a multicopter.
Source: (What is a MultiCopter and How Does it Work, n.d.)

1.6.2 Unmanned Ground Vehicles

An Unmanned Ground Vehicle is a mechanized equipment that moves across the surface of the ground and serves as a way for carrying or transporting something, but explicitly does not carry a human being (Gage, 1995). There are two types of ground vehicles: tracked and wheeled.

Tracked vehicles are better for off-road usages and carrying heavy loads, due to its reduced ground pressure and better traction. Although, building the suspension system for a tracked vehicle is more difficult than a wheeled robot suspension system. The suspension system has an important role related to vehicle's traction, by keeping the tracks or wheels on the terrain.

Wheeled vehicles attain faster road speeds offering the best solution when on-road usage exceeds off-road usage. Wheeled platforms allows less energy waste due to the reduced friction with the ground and reliability (to an extent), and operating and support costs are lower than tracked platforms. This makes wheeled platforms excellent candidates for support roles where overall mileage is high and primarily conducted on road (Hornback, 1998).

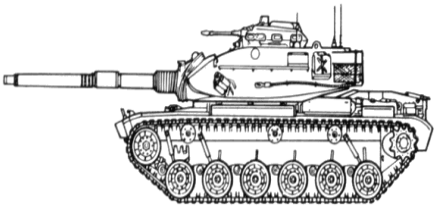


Figure 1-4 – Example of a tracked UGV.
Source: (Ata, 2014, p. 32).

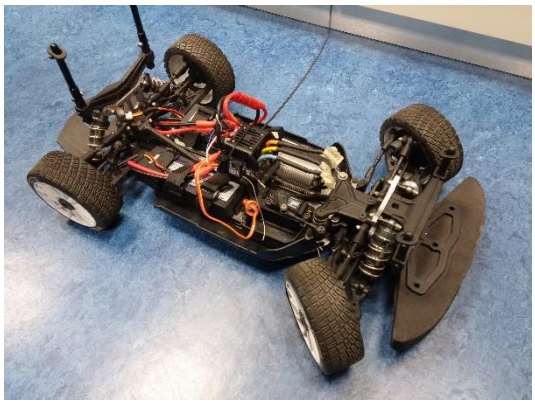


Figure 1-5 – Example of a wheeled UGV.

UGVs can have one of two steering modes, such as explicit and skid steering. Explicit steering works by changing the direction of the wheels, commonly used for wheeled vehicles and skid steering works by creating a differential velocity between the inner and outer wheels, commonly used for tracked vehicles (Shamah, 1999).

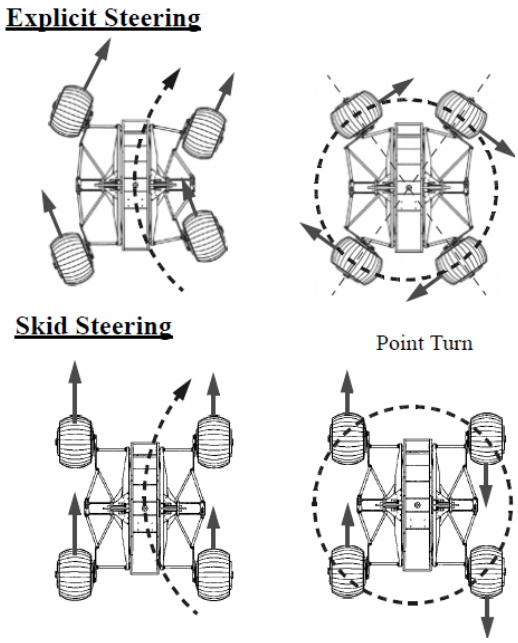


Figure 1-6 – Steering modes.
Source: (Shamah, 1999, p. 15).

1.6.3 Unmanned Surface Vehicles

This type of vehicles dates back to World War II. Only in the 1990s, a large proliferation of projects appears due to the technological progress and its increasing usability. As GPS have become more compact, effective and affordable, USVs have become more capable. An USV usually operate in one of the following three ways.

It can have a rudder and a propeller (Figure 1-7). When the propeller accelerates it creates a stream of water that propels the boat and pushes the water through the rudder. The rudder can move for both sides, diverting the water, causing the boat to rotate.

The other way a USV can work is by having two propellers (Figure 1-8) positioned at either side of the stern and rotating in opposite directions. Thrusting a propellant more than the other causes the boat to rotate (The effect of rudders, props and propwalk, 2012).

The third option is having a single motor fixed on a shaft able to rotate and redirecting the water flow to turn the boat.

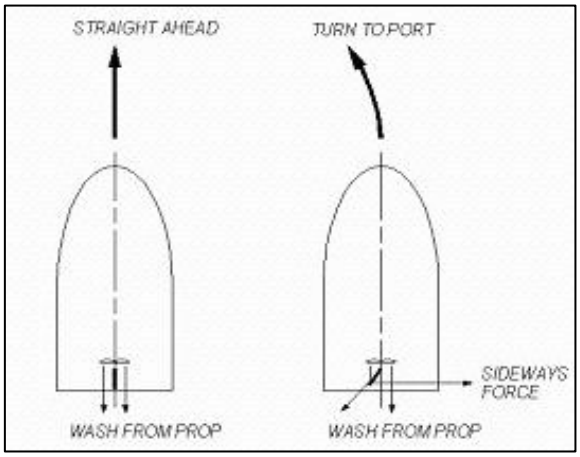


Figure 1-7 – Example of how a rudder works.
Source: (Do they both go round, mister?, n.d.).

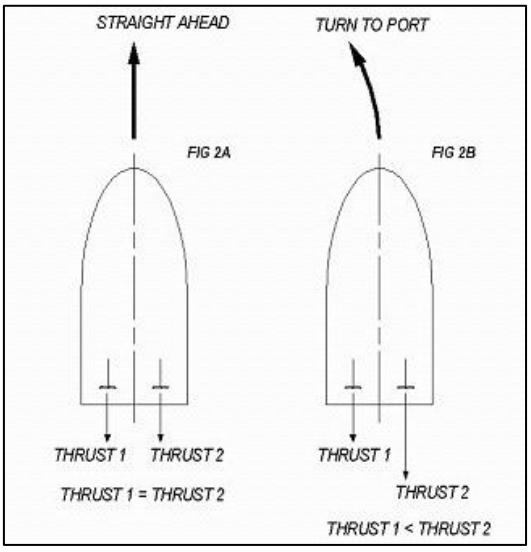


Figure 1-8 – Example of two propellers usage.
Source: (Do they both go round, mister?, n.d.).

1.6.4 Controller Board

Controller modules allow users to easily control various types of vehicles. Usually called flight controllers, they can have a compass, a gyroscope, a barometer pressure sensor, a magnetometer, an accelerometer and a GPS sensor. They are compact, lightweight, and powerful, allowing the user to turn any fixed-wing or multirotor vehicle (even cars and boats) into a fully autonomous vehicle. It connects to one radio receiver and interpret the commands sent by the user, controlling the power of each motor and making the vehicle move while holding it stable in the air. With GPS support, the controller is able to perform autonomous tasks, such as driving the vehicle to a certain latitude/longitude given by the user (APM 2.6 + Assembled, n.d.) (Pixhawk Autopilot, n.d.).

One of the most known controller boards for drones it's NAZA by DJI. NAZA-M V2 is capable of supporting nine types of multi-rotors configurations (quadcopters, hexacopters and octacopters) and enables the connection of a GPS module that allows the drone to maintain its position and travel across waypoints defined by the user using its intelligent orientation control. This board is commonly used with a radio controller but it's also capable of communicate with other DJI products like the Bluetooth module, which allows the user to connect a phone and control the drone by a mobile application or receive information such as the altitude, the velocity and battery status (NAZA-M V2, n.d.). However this board has a drawback for the purpose of this dissertation, the software is proprietary of DJI, only allowing to control copters.

Another of the most known controller boards is the Ardupilot Mega (APM). This controller module is based on Arduino mega platform and supports an open source firmware, capable of controlling fixed-wings, multirotors (tri, quad, hex and octocopter), helicopters and ground vehicles, depending on which firmware is loaded. This controller is able to take-off and land autonomously, control a gimbal (camera stabilizer), autonomous stabilization, navigate through waypoints (given by GPS coordinates) and telemetry (capable of sending useful data to the ground station, such as battery status). At the time being, there is an updated and improved version of the APM hardware, named Pixhawk. It has a 32bit architecture, a faster processor, more memory and other improvements. Pixhawk can run the same firmware as the APM, and has all the features of the APM, but with better accuracy and response times (Pixhawk Overview, n.d.).

An important aspect of this board is the possibility to connect a Raspberry Pi, and use it to transmit the parameters received by Pixhawk, or capture video and transmit it to a ground station (Communicating with Raspberry Pi via MAVLink, n.d.).

1.6.5 Firmware

The firmware for the APM or the Pixwawk, based on ardupilot software, has several branches, each one with a different purpose:

- ArduCopter: used to control multicopters, different configurations of copters (frame types) require different ArduCopter distributions;
- ArduPlane: for fixed-wings;
- ArduRover: to control ground vehicles and boats.

All of these three firmware uses a common messaging protocol called MAVLink and can run autonomous missions that are defined using a mission planning software able to communicate using MAVLink as well. (DIY Drones Firmware builds, n.d.)

The firmware is loaded from a Ground Control Station, such as MissionPlanner or QGroundControl (Loading Firmware onto Pixhawk/APM2.x/PX4, n.d.).

1.6.6 Ground Control Stations

A Ground Control Station (GCS) is a land or sea-based control center that provides the full control and monitoring of any UV vehicle connected.

1.6.6.1 Mission Planner

Mission Planner is a ground control station that runs on Windows, which is the one used in this dissertation. It is capable of loading a new firmware into the autopilot module to control another type of vehicle and if connected to the flight controller, it can show the information of the vehicle such as the altitude, actual position GPS coordinates, speed, etc. Mission Planner has a map in its layout where the user can give waypoints to send the vehicle once the GPS is connected. This software is also used to tweak the values and calibrate the vehicle. Mission Planner, uses MAVLink message protocol for communications (Mission Planner Overview, n.d.).

1.6.6.2 Andropilot

Andropilot is an open source android based GCS. It has gamepad support and a built-in touchscreen joystick for basic vehicle control, is able to set waypoints using a map view, edit the parameters and it can establish a User Datagram Protocol (UDP) and a Transmission Control Protocol (TCP) link, but in this case the vehicle has to be able to connect through Wi-Fi (Hester, 2013).

1.6.7 MAVLink

MAVLink is a communication protocol, which consists of a stream of bytes that are encoded by Mission Planner and sent to the APM, and vice-versa, via an USB serial or Telemetry connection. Encoding in MAVLink protocol requires arranging the package into a data structure and send it via the communication channel in a byte array, adding some error correction alongside (Balasubramanian, n.d.).

Each MAVLink message has the following informations:

- System ID, indicating the source of the message (i.e. Mission Planner);
- Component ID: the subsystem within the main system;
- Message ID: the type of message;
- Payload: the actual data.

1.6.8 Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It's capable of doing everything a desktop are expected to do, like browsing web or watching high-definition video (What is a Raspberry Pi, n.d.).

Raspberry Pi has a set of pins called General Purpose Input/Output (GPIO) pins that allow the connection of external sensors using Python or other programming languages.

It can be connected to the autopilot module through the UART0 pins. This way the module in the vehicle could communicate to the ground station not only by USB, but also through wireless networks (which require a dongle in the raspberry pi) (Communicating with Raspberry Pi via MAVLink, n.d.).

This page was intentionally left in blank

Chapter 2
FUNCTIONING OF AN UNMANNED VEHICLE

This chapter will explain the functioning of the three types of vehicles used in this dissertation (copters, cars and boats) as well as their components, in order to understand better how a 3D vehicle will work.

2.1 Components

Starting by the components of an electrical radio control power system, they break down into three main areas, the motors, the ESCs and the battery. It will also be explained the base functioning of a radio control system.

2.1.1 Motor

There are a wide variety of available motors which have to be chosen according to the type of vehicle, its dimensions and purpose.

In this dissertation it will be used brushless DC motors, since they are lighter compared to brushed motors with the same power output, and they have a longer life span due to the fact that the brushes in brushed motors wear out with the time and may cause sparking.

There are two types of motor designs, outrunner motors and inrunner motors. In outrunner motors the can moves with the shaft and in inrunners the can stay still while the shaft spins in the middle.

Some motors are labeled with numbers that need to be explained.

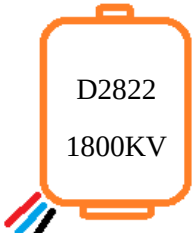


Figure 2-1 – Motor label example.

Labeled as D28 in Figure 2-1 is the motor or rotor diameter that can be the physical width of the can or the physical width of the rotor inside the can. Commonly, the wider a motor is, the more torque will generate. Number 22 is the motor or rotor height.

The last number, indicated as 1800KV is the revolutions per minute per volt (rpm/V). That means that for every volt that is supplied to the motor in a no load condition, how many revolutions it will turn per minute. In this example the motor will turn 1800 revolutions every minute for every volt applied. (Büchi, 2012)

There are other key metrics which should be taken into account when choosing a motor.

Power, P is given by $P = U \times I$, where U and I are voltage and current in volt and ampere, respectively. The motor will be more powerful for higher values of P.

Thrust (for aerial vehicles) shows how much thrust the motor produces. The thrust depend on the propellers attached, or the battery used.

The voltage that the motor needs will affect the choice of the batteries. Each battery setup will have a recommended propeller (in case of being an aerial vehicle).

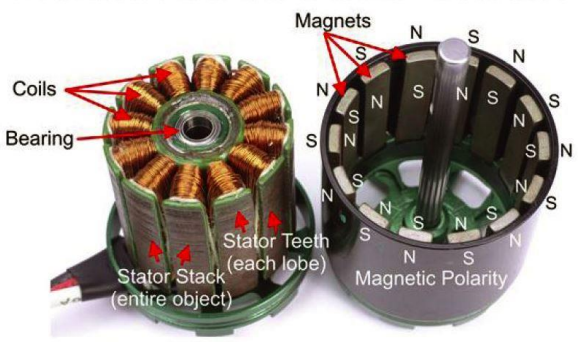


Figure 2-2 – Outrunner components.
Source: (Brushless motors - how they work and what the numbers mean, 2014).

The motors used have three input wires, Figure 2-2 shows the basic principle of how an outrunner brushless motor works. The motor have permanent magnets fixed in the can / rotor (rotating part) and the stator (stationary part) has a coil arrangement as shown (the number of poles may vary according to the motor). By applying current to the coil, it will energize and become an electromagnet that will pull the magnet with the opposite polarity toward it, then it will energize the next coil and de-energize the previous one, and the magnet will continue to rotate as this process is repeated, in Figure 2-3 the process will go from A, to B, to C and repeat (Brushless DC Motor, How it works ?, 2016) (Büchi, 2012).

Each wire is connected to a set of coils and the electronic speed controller (ESC) is responsible for triggering them. The frequency at which this process is done will dictate the velocity of the motor.

Increasing the number of poles gives higher torque but will reduce the maximum speed (Mohammed, 2014, pp. 6-17).

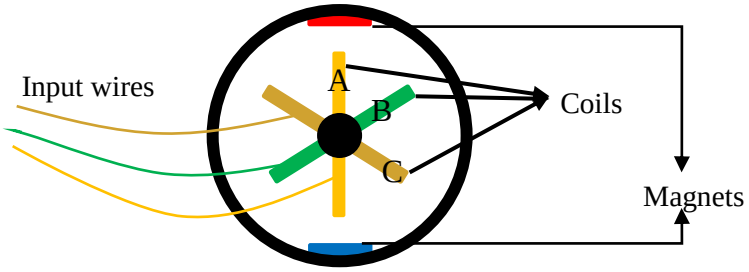


Figure 2-3 – Outrunner motor design.

2.1.2 Servo Motor

The servos used are analog and they are made up of small brushed motors, a potentiometer, and a couple of gears. The potentiometer regulates the movement of the motor and allows the servo to move an exact amount of degrees. This way servo motors are able to provide precise control and accommodate complex motion patterns and profiles more readily. They can also operate at zero speed while maintaining enough torque to maintain a load in a given position. The gears are for gear reduction because the motor in a servo has a higher rpm but not much torque, the gears takes the rpm down and creates more torque. Most servos, unlike conventional electric motors, do not move in continuous rotations. The standard servo moves anywhere between 0 and 180 degrees. (Murilhas, 2015)

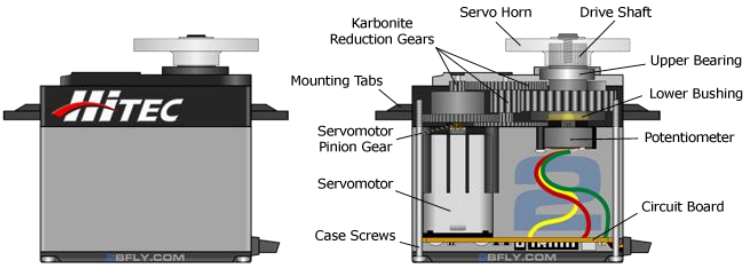


Figure 2-4 – Servo anatomy.
Source: (Servo Anatomy, n.d.)

The servo has also a connection cable composed by three wires, red for power, black for ground and a white one for the signal. The cable connects to the controller or the radio receiver that uses Pulse Width Modulation (PWM) to send pulses of electricity to the servo motor to turn in the direction and distance requested by the input controls. Due to this characteristics servos are used for the car's steering and in the rudders of planes and boats, to have a fast and precise motion.

2.1.3 Electronic Speed Controller

Brushless motors are three phase motors so each of the three output wires on the ESC connects to each one of the input wires on the motor. If two of the connected wires are switched, the motor will spin in the other direction. On the other end, there are the ESC's inputs that will be connected to the battery or the power distribution board in case of being a multicopter, and another cable that will be connected to the controller board, that can supply the 5V necessary for the board to work but also has a signal cable (white) and the voltage level on that signal wire, sent by the controller, will make the ESC change the frequency of the pulses sent to the motor coils, and change the motor's speed.

Figure 2-5 – Example of the ESC connections.

A lot of ESCs, including the ones used in this dissertation, comes with battery eliminator circuits (BEC) on them that provides the 5V that runs the controller board, the servos and everything else on the model. ESCs have limited current that can support and pass to the motors. In Figure 2-5, the maximum throughput is 20A, which is a factor of choice because the ESCs must support currents higher than the maximums currents that the motor is able to draw.

2.1.4 Battery

The batteries used in the vehicles for this dissertation will be Lithium Polymer (LiPo) batteries. LiPo batteries are rechargeable batteries that uses Lithium-ion technology. They have three main advantages when compared to the other common types of batteries, Nickel-Metal Hydride (NiMH) or Nickel Cadmium (NiCd) batteries. They are much light weighted, offering much higher capacities, allowing them to hold more power and offer higher discharge rates. They also have three drawbacks, such as shorter lifespans, when punctured they can catch fire due to its sensitive chemistry and they require more care when they are charged, discharged and stored.

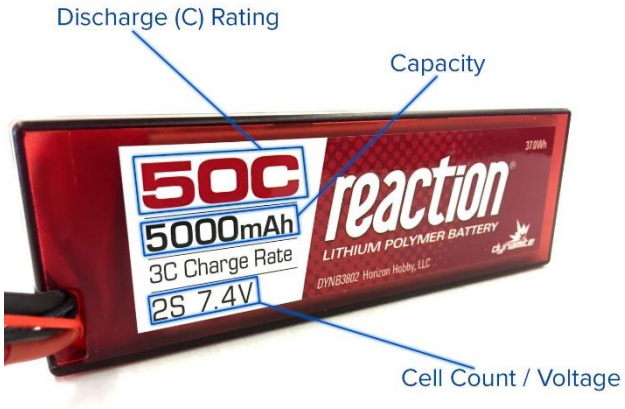


Figure 2-6 – 5000mAh battery example.
Source: (A Guide to Understanding LiPo Batteries, 2016).

There are three numbers that should be taken into account when choosing the right battery.

The **Voltage**, corresponding to the voltage of the battery at nominal charge, in Figure 2-6 that voltage is 7.4V because the pack has 2 LiPo cells connected in series, which is indicated by the 2S, each battery cell has a nominal voltage of 3.7V. Each cell has 4.2V when they are fully charged. When the battery pack is for example a 2S2P that means the pack has two cells connected in series and two connected in parallel. The voltage will determine the motor rpm (as described on section 2.1.1), so batteries with more cells will increase the vehicle maximum speed.

The **Capacity** of the battery, indicated by the mAh (milliamps/hour), a measure of how much power the battery can hold. Indicating how much drain can be put on the battery to discharge it in one hour. In Figure 2-6 the capacity is 5000mAh so that means if 5000 milliamps (5 amps) are being drawn for an hour, the battery will be fully discharged at that time.

The **Discharge Rating**, indicated by the 50C in Figure 2-6 , is how fast the battery can be discharged safely and without harming the battery, in this case 50 multiplied by the capacity. The discharge rating in this example is calculated by the following formula:

$$50C = 50 \times capacity = 50 \times 5(A) = 250A \qquad (2.1)$$

250A is the maximum current that can be pulled from the battery safely. More than that will result in the degradation of the battery at a faster pace or it could burst into flames.

The battery in the example also has a charge rate of 3C, which means that can be charged at a speed, three times his capacity (3x5(A) = 15A). Some batteries don't have this information, so they should be charged at a maximum of 1C (Murilhas, 2015, pp. 32-33).

2.1.5 Radio Control System

One of the most common ways to control an UV is by Radio Control (RC). It's cheap and easy to set up. The radio control module has two parts, the transmitter and the receiver. The transmitter stays with the user and translates the movement of the joysticks to electric signals to send them to the receiver.

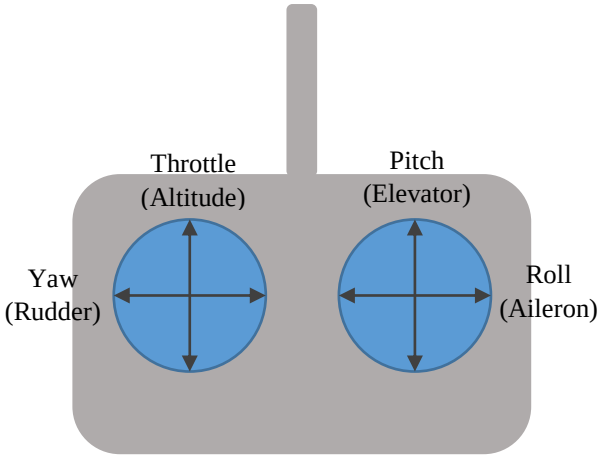


Figure 2-7 – Transmitter mode 2 joystick functions.

A typical RC transmitter has about 4 to 6 channels. The Figure 2-7 shows a four channel RC transmitter with two joysticks, each axis is a channel. This example shows a configuration having the throttle on the left and the pitch on the right (known as mode two configuration). The right joystick is self-centered in both axis and the left is self-centered in the X axis but not in the Y axis to allow to maintain the throttle position (Rönnberg, 2004).

The receiver stays in the vehicle and has to be tuned on the transmitter frequency to receive the signal for each channel of the transmitter. It can be connected to a servo motor, an ESC or to a controller board.

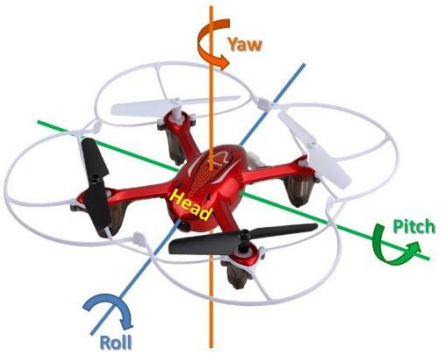


Figure 2-8 – Copter axis.
Source: (What are all those Knobs and Buttons on Your Drone's Transmitter Used for?, 2016).

2.2 Construction of an Unmanned Ground Vehicle

For the purpose of this dissertation a Basher 1/8 Scale Rally was used to simulate an UGV vehicle. This model is well-equipped with 17mm shocks, a 2100KV motor, a 4S-rated 120A speed controller and a servo motor to control the steering. It also comes with a radio receiver and transmitter to control the vehicle. It was only necessary to choose the batteries, which in this case the recommended batteries applied were two 5000mAh 2S 7.4V 30C batteries connected in series.

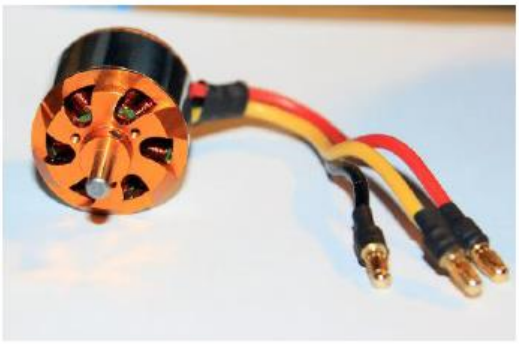
Figure 2-10 – Basher 1/8 scale rally interior.
Source: (Vieira, 2014)

Figure 2-9 – Turnigy 5000mAh 2S 7.4V 30C hardcase pack.
Source: (HobbyKing, n.d.)

In this case the car can be put into operation without a controller board. The ESC is directly connected to one of the receiver outputs, in the channel corresponding to the throttle axis (Figure 2-7) and the servo motor to the channel corresponding to the roll. The receiver and servo are powered due to the BEC in the ESC that provides the 5V needed to work. The controller board will be installed later on.

2.3 Construction of an Unmanned Aerial Vehicle

To simulate an UAV, a hexacopter was built. This type of copter was chosen to carry more weight and have more stability. Starting with the motors, six Turnigy D2836/8 1100KV Brushless Outrunner Motors were used (*Figure 2-11*).



Turnigy D2836/8 1100KV Brushless Outrunner Motor

Spec.
Battery: **2~4 Cell /7.4~14.8V**
RPM: **1100kv**
Max current: **18A**
No load current: **1A**
Max power: **336W**
Internal resistance: **0.107 ohm**
Weight: **70g (including connectors)**
Diameter of shaft: **4mm**
Dimensions: **28x36m**
Prop size: **7.4V/11x7 14.85V/7x3**
Max thrust: **1130g**

Figure 2-11 – Specs of the motors used to construct the UAV.
Source: (Turnigy D2836 / 8 1100KV Brushless Outrunner Motor, n.d.)

This type of motors are well ventilated to reduce heating. When choosing the motors it must be ensured that the copter will be able to fly easily. The drone must start flying as soon as the joystick passes 50% of the throttle power. This means that the motors are functioning at half the power. One simple rule to ensure that the motors are selected properly is to do the following calculation:

$$Propulsion\ per\ Motor = \frac{(weight\ of\ copter \times 2)}{Number\ of\ Motors} \quad (2.2)$$

To know the supported weight for the copter, using the formula above and the motors shown in *Figure 2-11* as an example, 1.130kg as the propulsion per motor and 6 as the number of motors, the copter can weight 3.39 kg.

Since the motor draws 18A, the ESCs used were the "Hobbyking 30A ESC" with a 3A UBEC to power the controller board. The 30A gives a good margin so the motors can work at full power without causing much effort to the ESCs. The propellers used were

10x4.7 that are the ones recommended for this type of motors. To power the copter a "ZIPPY Compact 5800mAh 4S 25C Lipo Pack battery" was used. This was based on the motors specs, as shown in Figure 2-11. The batteries that can be used are from 2S to 4S. The 5800mAh at 25C allow a discharge rate of 145A (section 2.1.4), higher than all the motors at full power, which draw a total of 108A. This assures the battery can work safely even with the motors at full power. The frame chosen was a "Turnigy H.A.L. (Heavy Aerial Lift) Hexcopter Frame 775mm", this frame weights 1380g which is heavy for a hexacopter frame but allows the drone to remain stable, even with wind gusts. The total weigh of the constructed hexacopter is about 3,20kg which is less than the 3,39kg calculated.

A quadcopter was also built for testing purposes as can be seen in Annex B, Figure B. 2.

Figure 2-12 - Hexacopter built.

2.4 Unmanned Surface Vehicle

For the purpose of this dissertation and taking into account the capabilities of the hardware used, the construction of a USV wasn't necessary due to the controller board using the same firmware to control both a boat and a car. In case of being a boat with two parallel motors, its function will be similar to a tank, which is a mode of the rover firmware in the controller board. In this mode the controller board changes the power of each motor individually to turn the vehicle. If it's a rudder boat or with a turning motor, it will be similar to the steering of a car, there is a motor to move the vehicle and a servo to change its direction that will turn the steering or the rudder. Since the firmware used is the same for both the car and the boat, the application works with the rover firmware for

both vehicles. However it will need to have an option to invert the controls due to the fact that the boat has the servo in the back facing backwards, so the controls will be reverse.

2.5 3D Vehicle

The 3D vehicle, titled as such because it's able to move in three different environments: land, air and water, being a combination between UGV, USV and UAV. The mobile application to control a 3D vehicle and change between modes requires to have or simulate a 3D vehicle to test the application. To build a 3D vehicle it was necessary to create a custom vehicle able to deal with the different scenarios. Although, this would take a lot of time and effort to be constructed and is not the scope of this dissertation, since this will be focused in the application development. For this reason a 3D vehicle simulation was chosen. To do this, three types of vehicles were tested: a car to simulate a ground vehicle, a boat to simulate a surface vehicle and a copter to simulate an UAV.

The application was developed to deal with any of these three types at the same time. This allows the user to connect a ground, a surface and an aerial vehicle simultaneously and change between their controls, simulating a change between the different modes in the 3D vehicle. If the frame is built in a future work, it will only be necessary to put together all the hardware in these three types of vehicles, such as the motors, ESCs and the remaining electronics in the same frame and connect them concurrently to the application and the vehicle will be ready to control.

The application being able to deal with any type of vehicle has an advantage against only being able to deal with 3D vehicles. This way a user can connect any kind of vehicle and control it, even if it's only a vehicle at a time.

This page was intentionally left in blank

Chapter 3
CONTROLLER SYSTEM

In this chapter will be explained the different controller boards used in the vehicles and their purpose.

Controller boards for UVs are generally referred as flight controllers because they are mostly used for aerial vehicles, responsible for managing the flight of multi-rotor aircrafts and airplanes. Its purpose is to stabilize the aircraft during flight and to do this it takes the signals from on-board gyroscopes and passes them to the processor, which processes signals according to the user selected firmware (corresponding to its type of vehicle) and passes the control signals to the installed ESCs to make fine adjustments to the motors rotational speed which in turn stabilizes the aircraft. Controller boards also use signals from radio and other sources like Bluetooth or Wi-Fi modules and relays these signals (e.g. aileron, elevator, throttle and rudder) together with stabilization signals to the processor, which once processed send them to the ESCs to turn and control the vehicle orientation.

There are several controller boards available in the market, although, a controller board with advance functionalities like GPS and auto guide was necessary. One of the most known flight controllers is Ardupilot Mega (APM) which was used in ground vehicles, and its evolution, Pixhawk, with better performance, which was chosen as the main controller to be used in aerial vehicles since they have more motors and require more processing power due to all the calculations the controller has to do in order to maintain the vehicle stable in the air.

3.1 Pixhawk

Pixhawk Autopilot is one of the most known flight controllers and uses PX4 hardware. It has high performance and is suitable for most types of RC vehicles, fixed wings, copters, cars and boats, depending on the firmware loaded. It has a variety of inboard sensors to guide and maintain the vehicle stable such as an Invensense MPU 6000 3-axis accelerometer/gyroscope, ST Micro L3GD20H 16 bit gyroscope, LSM303D 14 bit accelerometer / magnetometer and a MEAS MS5611 barometer. This controller board is also able to connect an external GPS which allows the vehicle to drive autonomously and requires a micro SD card to store the datalogging. Pixhawk has a 32bit STM32F427 Cortex M4 core processor, 168 MHz, 256 KB of RAM and a second processor, STM32F103, to serve as a backup failsafe to provide manual recovery with its own power supply if the main one fails. This gives the controller board incredible performance and reliability (Pixhawk Autopilot, n.d.).

To communicate with external systems such as ground control stations this controller board uses the MAVLink messaging protocol.

3.1.1 Connections

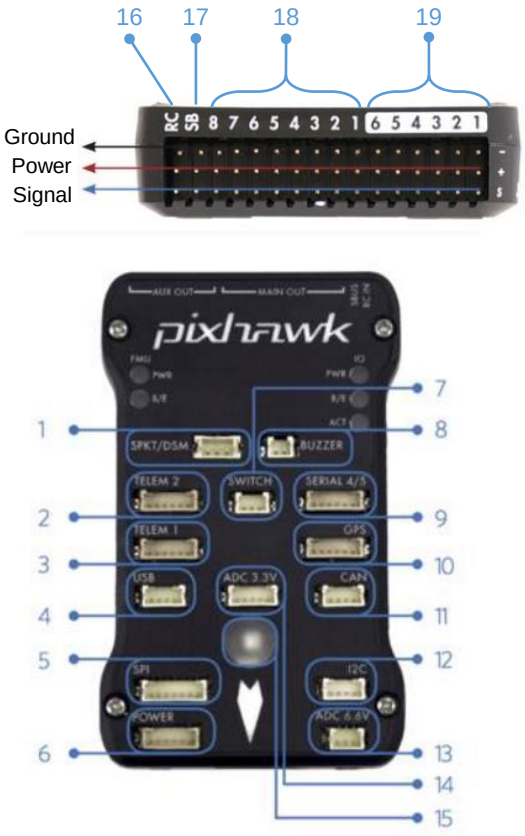


Figure 3-1 - Pixhawk connector assigments.
Adapted from: (Pixhawk Overview, n.d.).

Nº	Description
1	Spektrum DSM Receiver
2	Telemetry (on-screen display)
3	Telemetry (radio telemetry)
4	USB
5	SPI (serial peripheral interface) bus
6	Power module
7	Safety switch button
8	Buzzer
9	Serial
10	GPS module
11	CAN (controller area network) bus
12	I^2C splitter or compass module
13	Analog to digital converter 6.6V
14	Analog to digital converter 3.3V
15	LED indicator
16	Radio control receiver input
17	S.Bus output
18	Main outputs
19	Auxiliary outputs

Table 3-1 – Pixhawk port description.

Pixhawk also has a reset button on its side, a SD card slot and a Micro-USB port.
The ESCs from the motors are connected to the main output port according to the type of vehicle.

For the car (rover mode) the output pins are the pin 3 to the motor (throttle) and pin 1 for the servo (steering). The boat is also pin 1 and pin 3 but in this case are both to the motors.

Since the boat is a two parallel propeller boat, it will use the rover mode but configured to skid steering. To do this a parameter called *SKID_STEER_OUT* and *SKID_STEER_IN* must be changed to "1" in the GCS (Pixhawk Wiring Quick Start, n.d.).

3.1.2 Connecting the GPS

The *3DR UBlox GPS + Compass module* is the recommended GPS sensor for the APM and the Pixhawk, which was the one used in this dissertation. The GPS is required for autonomous drive and for better stabilization. Some flight modes such as 'auto', 'loiter' and 'return-to-launch' requires GPS lock.

Figure 3-2 - 3DR UBlox GPS + compass module.
Source: (Pixhawk Wiring Quick Start, n.d.).

The GPS port is connected to the Pixhawk through a 6-wire cable and the MAG port is connected to the I^2C port in Pixhawk using a 4-wire cable. The external magnetometer is used to reduce the error caused by the electric interference from the ESCs.

It's recommended to mount the GPS+compass module outside the vehicle and in an elevated position, as far as possible from the motors and ESCs with the arrow facing forward, due to the previously mentioned interference. It's also recommended to twist the wires and to put the module at least 10cm away from the batteries and their wires.

After installing the GPS + compass module it's necessary to calibrate it, which should be done on the surface level. The calibration is done with the help of Mission Planner software. When the GPS is turned on, two LEDs should light up, a red one to show the GPS has power and a flashing blue one to indicate that the GPS has successfully locked.

3.1.3 LEDs

Knowing the LED's meaning is important to know the vehicle state and problems.

LED behavior	**Description**
Blinking Red and Blue	Gyroscopes initializing.
Blinking Blue	Vehicle is disarmed and with no GPS lock.
Solid Blue	Armed but without GPS lock.
Flashing Green	Disarmed with GPS lock
Solid Green	Armed with GPS lock acquired. Ready to fly.
Double Yellow Blinking	Failing pre-arm checks (systems will not arm)
Single Yellow Blinking	Radio failsafe activated
Blinking Yellow	Battery failsafe activated
Blinking Blue and Yellow	GPS glitch or GPS failsafe activated
Blinking Purple and Yellow	Barometer glitch
Solid Red	Error

Table 3-2 – Pixhawk LEDs behavior meaning.

3.1.4 Powering

The Pixhawk is powered by a *3DR* Power Module. The power module is connected between the distribution board or the ESCs and the battery and connected to the Pixhawk with a 4 wire cable. It will be used to supply 2.25A at 5.37V (up to 18 volts and 90A maximum). The vehicle cannot draw more than 90A from the battery or else the power module can overheat and fail. This allows the Pixhawk to read the battery voltage and calculate the battery remaining.

3.2 Ardupilot Mega

The Ardupilot Mega (APM) is another of the most known flight controllers among the hobbyist community. APM is a pro-quality FMU autopilot based on Arduino Mega platform, which can turn any RC vehicle into a fully autonomous unmanned vehicle. It has several inboard sensors such as a barometric pressure sensor for altitude reading, 3-axis magnetometer (compass), 3-axis accelerometer and 3-axis gyroscope, internal memory to store data logging, 2 way telemetry and able to attach an external GPS module. It requires no assembly and is ready for firmware, which will change according to the type of vehicle (helicopter, fixed-wing, multicopter, ground rovers and boats). With

information from its sensors the controller is able to handle both autonomous stabilization and autonomous navigation through GPS waypoints. (APM 2.5 and 2.6 Overview, n.d.)

To communicate with external systems this controller also uses MAVLink.

3.2.1 Pinout

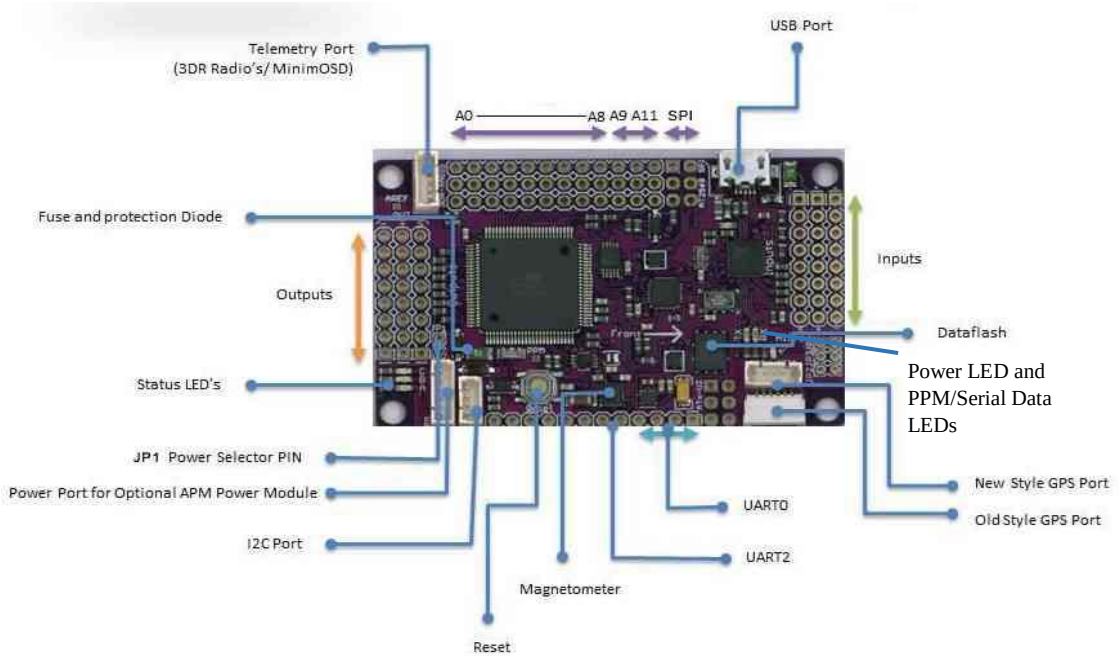


Figure 3-3 – APM 2.5 Board Features.
Adapted from: (APM 2.5 and 2.6 Overview, n.d.).

Pin Nº	Function
A0	Sonar
A1	Voltage
A2	Current
A3	Optical Flow
A4	Motor Led
A5	Motor Led or Beeper
A6	Motor Led or GPS Led
A7	Motor Led or Arm Led
A8	Motor Led or RSSI
A9	Motor Led or Camera Shutter
A10	Camera Roll
A11	Camera Pitch

Table 3-3 – A0 to A11 pins functions.

A0 to A8 pins in Figure 3-3 are analog input pins. These pins take up to 5V and are generally used for sonars, airspeed sensors and led indicators to show if the GPS has found satellites or if the drone is armed and ready to fly. A9 is usually used to trigger the camera shutter, A10 to connect the servo controlling the camera roll and A11 to connect the servo controlling the camera pitch, which together can stabilize the camera in both axis.

APM has 8 input pins. Each pin connects to one of the channels in the radio receiver. (Connecting the Radio Receiver (APM2), n.d.)

Input Pin Nº	**Function**
1	Roll / Aileron
2	Pitch / Elevator
3	Throttle
4	Yaw / Rudder
5	Aux 1 (mode switch)
6	Aux 2 (optional)
7	Aux 3 (optional)
8	Aux 4 (optional)

Table 3-4 – APM2 input pins connections.

There is also 8 output pins in APM that have different functions according to the firmware installed. If the ArduCopter firmware is installed, each pin connects to an ESC controlling one motor, up to 8 ESCs (octocopter). If the ArduRover firmware is installed, there are two possibilities, if skid steering is disabled (for cars and boats with rudders), pin 1 connects to the servo controlling the steering, and pin 3 connects to the ESC controlling the motor. If skid steering is enabled (for tanks and boats with two parallel motors), pin 1 connects to the ESCs controlling the left motor and pin 3 connects to the ESC controlling the right motor (Rover APM2.x Wiring and Quick Start, n.d.).

Output Pin Nº	**Multicopter**	**Cars and Boats**	**Skid Steer**
1	Motor 1	Servo	Left motor
2	Motor 2		
3	Motor 3	Motor/ESC	Right motor
4	Motor 4		
5	Motor 5		
6	Motor 6		
7	Motor 7		
8	Motor 8		

Table 3-5 – APM 2 output pin connections.

3.2.2 Connecting the GPS

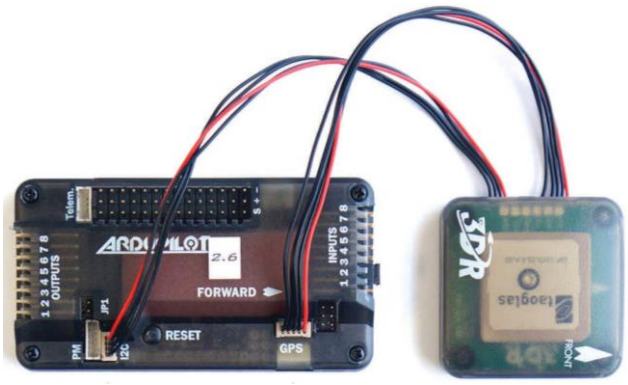


Figure 3-4 – APM 2.6 + GPS module.
Adapted from: (3DR UBlox GPS + Compass Module, n.d.).

The difference between the APM 2.6 and its previous version APM 2.5 is in the way the GPS module is connected. APM 2.5 has an in-board magnetometer (for compass) while APM 2.6 uses the magnetometer in the GPS module. This allows to mount the compass away from the interferences caused by the ESCs. For this reason is recommended to cut the connector to the internal compass in APM 2.5 and use the one in the GPS module (Rover APM2.x Wiring and Quick Start, n.d.).

To connect the APM 2.6 to the *3DR UBlox GPS module* it's only necessary to attach the *new GPS style port* in the APM to the GPS port in the GPS module with a 5-to-6 wire cable and the I^2C serial bus port in the APM to the MAG (magnetometer) port in the GPS module with a 4 wire cable.

3.2.3 LEDs

APM has some LEDs that help to understand the state of the board. There are three status LEDs (Figure 3-3) labelled A, B and C. Starting by the A led (power led), at the bottom, this led is red and has three states, single blinking, that means the vehicle is disabled and motors will not spin. Double blinking indicates that the vehicle is disarmed and had a failure in pre-arm checks which prevents the vehicle to be able to arm. When A LED is solid red that means the vehicle is armed and motors will spin when throttle is raised. B LED, above A LED as shown in Figure 3-3 is a yellow LED and only flashes during calibration or as part of the in-flight auto trim feature. The top LED, labelled C, is blue and indicates the GPS state. When the LED is off there is no GPS attached, when is

blinking the GPS is working but has no lock (still searching for satellites) and when is solid blue the GPS is locked. (LEDs (APM 2.x), n.d.)

3.2.4 Powering

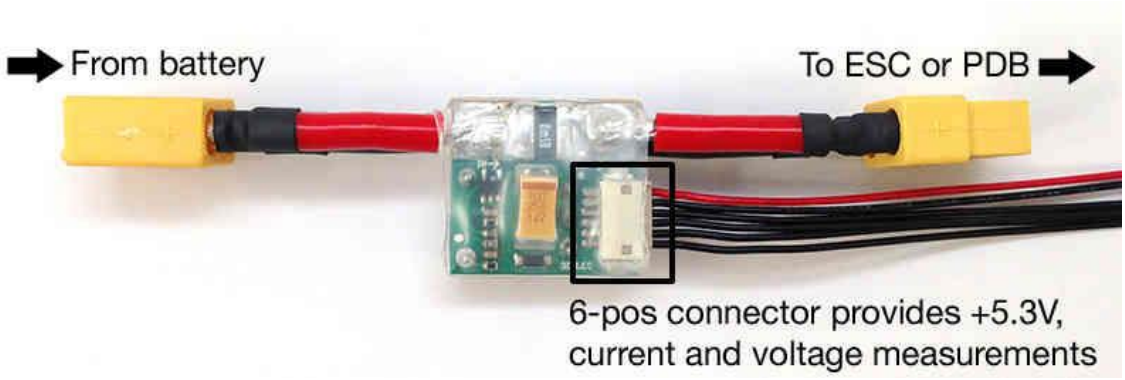


Figure 3-5 – 3DR power module.
Source: (3DR Power Module, n.d.).

The most common way to power APM is to connect a 3DR Power Module which supplies 2.25A at 5.37V converted from the power coming from the battery (up to 18 volts and 90A maximum). The power module is connected in the power port for optional power module in Figure 3-3 with a 6-wired cable. The APM is then able to power the radio receiver or some other peripheral devices such as the Bluetooth module or a sonar that can be attached. However, servos cannot be fed from the APM, they have to be powered separately, APM serves only to control them. APM functions at 5V however there is an input voltage of 5.37V because there is a Zener diode that has a loss of 0.37V (3DR Power Module, n.d.).

One of the main advantages of the 3DR Power Module is that allows the APM to be able to monitor the battery's voltage and current and trigger safety mechanisms when the voltage is too low and allows the autopilot firmware to compensate more accurately the interference on the compass from other components.

It's also possible to power the APM with the 5V BEC in the ESCs by putting a jumper in the JP1 Power Selection Pin (Figure 3-3) to connect the power circuit coming from the output pins. After putting the jumper it's only necessary to ensure that the power and ground cables coming from one of the ESCs are connected. Only one ESC is necessary to provide the power to the APM so the power and ground cables from the remaining ESCs can be cut, leaving only the signal wire (white wire) for its control. The BEC needs

to have 5V and at least 2 amps. However if there are servos connected to the output pins the jumper should be removed to separate the circuit coming from the output pins from the rest of the APM because the servos draw too much current and can burnout the APM. In this case the APM should be powered by connecting a power and ground cable coming from one of the ESCs to one of the input ports. (Powering the APM2, n.d.)

3.3 Raspberry Pi

To connect the vehicle to the network its necessary to have a way to forward the controller board messages through Wi-Fi or 3G/4G. To do so it will be used a Raspberry Pi (RPi) as shown in *Figure 3-6*.

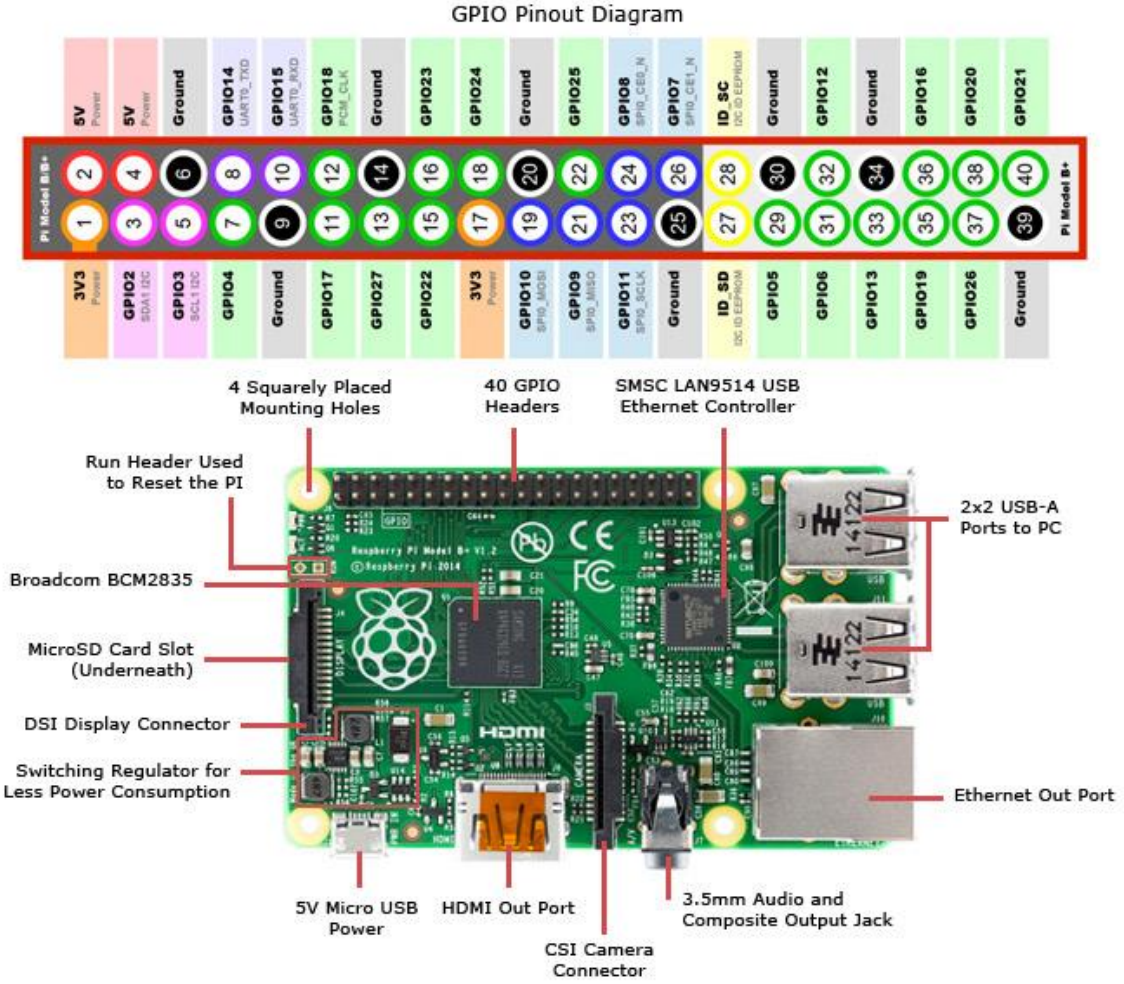


Figure 3-6 – Raspberry Pi 2 model B pinout and circuit notes.
Source: (Raspberry Pi Pinout Diagram | Circuit Notes, n.d.).

Raspberry Pi is a tiny and affordable computer, almost the size of a credit card. It's able to do everything a normal computer does such as surf the internet, play videos in high definition, run programs and play games, although is slower than modern laptops or desktops, but is still very useful to do projects that require more than a basic microcontroller (such as Arduino devices). Raspberry Pi runs on Linux operating system and is an open source hardware computer, with the exception of the primary chip the Broadcomm SoC (System on a Chip). Raspberry Pi has even the capability to connect to external devices by using GPIO pins, USB ports, HDMI port, 3.5mm audio jack and has an SD card slot.

The model used in this dissertation was Raspberry Pi 2 model B, the second generation of Raspberry Pi 2. The difference to the previous version (Pi 1 model B+) is that has a 900MHz quad-core ARM Cortex-A7 CPU and 1GB of RAM. The Raspberry Pi is powered by the USB interface at a minimum of 3.3V and maximum of 5V. (Raspberry Pi 2 Model B, n.d.)

The most common Linux operating system distribution to install in the Raspberry Pi is Raspbian which is based on Debian and it has different versions. For the purpose of this dissertation the version installed was Jessie. An internet connection will be needed to update and install the required programs so it can either be connected by an Ethernet cable or a Wi-Fi dongle.

3.3.1 Camera

Raspberry Pi has two official cameras modules, the camera module and the Pi NoIR camera. Camera module has two versions, the first one has a 5-megapixel OmniVision OV5647 sensor and the second version has a Sony IMX219 8-megapixel sensor. It can film in high definition and supports 1080p30, 720p60 and VGA90 video modes, as well as still capture. To attach the camera it's used a 15cm ribbon cable to the CSI camera connector in Raspberry Pi (Figure 3-6).(Camera Module V2, n.d.) The Pi NoIR is na infra-red camera and it also has two versions, version one. 5-megapixel OmniVision OV5647 sensor and version two has a Sony IMX219 8-megapixel sensor. It offers everything the regular camera module offers but does not have an infra-red filter (NoIR = No infrared) which means that is able to see in the dark with infrared lighting. It connects the same way as the regular camera module. Both cameras work with all models of Raspberry Pi and it can be acessed through the MMAL and V4L APIs (Pi NoIR camera

V2, n.d.). The Raspberry Pi also suports standard USB webcams to take pictures and videos, however the quality of the camera module is highly superior, which was the one used in this dissertation (Using a Standart USB Webcam, n.d.).

After connecting the camera, there is a needed to install Video4Linux library which is a collection of drivers and an API for supporting realtime video capture on Linux systems. To install it the following command was inserted into the console:

sudo aptitude install libv4l-0

Not every USB webcams work with the Raspberry Pi. If a USB webcam was used it can be tested by installing fswebcam. Starting by its instalation, the following command should be inserted:

sudo apt-get install fswebcam

Next it's required to enter the command *fswebcam* followed by the filename, which will take a picture using the webcam, and save it to the filename specified. Example:

fswebcam test.jpg

The image should be taken without errors and saved into */home/pi* directory by default (Using a Standart USB Webcam, n.d.).

3.3.2 Powering

To power the Raspberry Pi placed in the vehicle it was used a 3A/5V UBEC that is connected in parallel with the cables that go from the battery to the ESC's. This UBEC will transform the voltage coming from the battery to the 5V needed to power the Raspberry Pi.

3.4 Connecting Raspberry Pi to the Pixhawk

To connect the Pixhawk to Raspberry Pi it was used the USB port in the Raspberry Pi and the micro USB port in the controller board. This is the easier way to connect. It can also be connected via serial by connecting the telemetry port in the Pixhawk to the GPIO pins in the Raspberry Pi as shown in figure Figure 3-7.

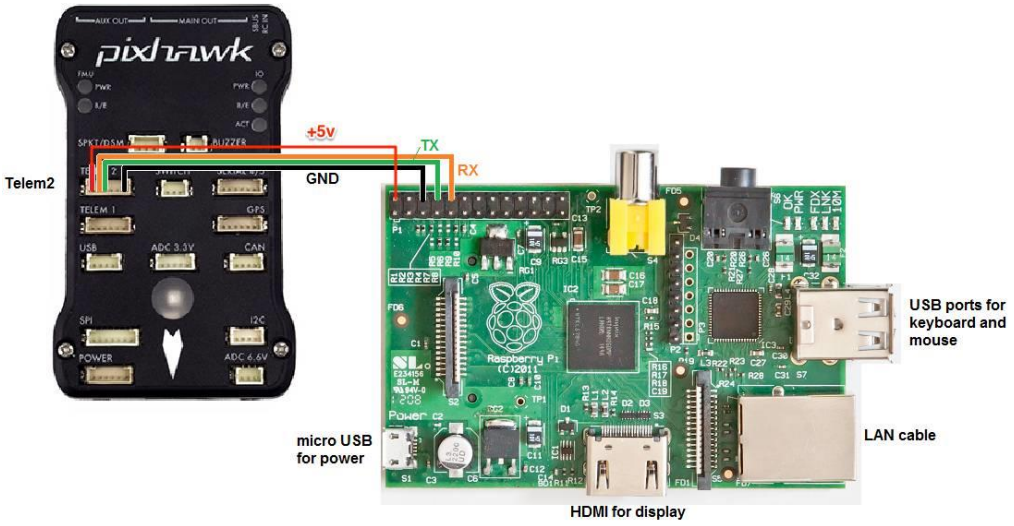


Figure 3-7 – Raspberry Pi to Pixhawk connection.
Source: (Communicating with Raspberry Pi via MAVLink, n.d.)

The version of the Raspberry Pi used in this dissertation is different from the one shown in Figure 3-7. Based on the pins number in Figure 3-6, the 5V cable is connected to the pin 2, ground cable is connected to pin 6, RX is to pin 10 and TX is to pin 8.

3.5 Connecting Raspberry Pi to the APM

One of the main advantages of the APM for the purpose of this dissertation is the ability to connect to the Raspberry Pi. To link them is only necessary to attach three simple connectors as shown in Figure 3-8. The positive wire cannot be connected as this powers the Raspberry Pi but does not supply enough power for the 3G USB network interface. The other reason for not connecting the power cable is that if the Raspberry Pi has a kernel malfunction or some other issue it can reboot the APM and all communications will be lost. (NCC Group Publication, 2016)

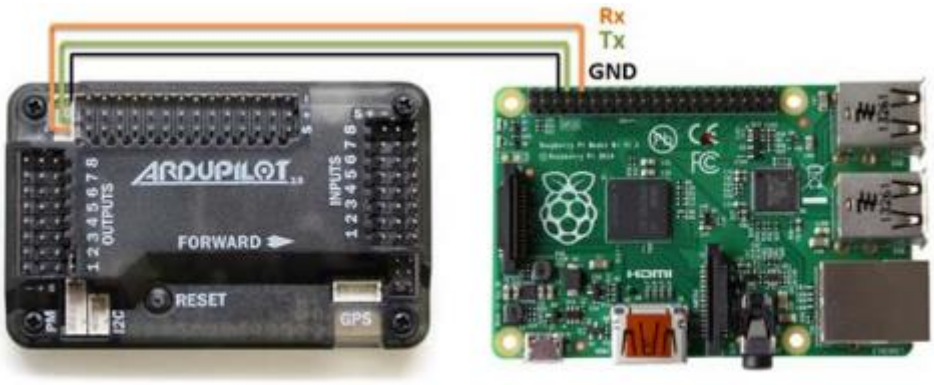


Figure 3-8 - APM to Raspberry Pi connection.
Source: (NCC Group Publication, 2016).

Chapter 4
SOFTWARE TOOLS AND FRAMEWORKS

This chapter will explain how to arrange the software components and which programs were installed in order to communicate with the mobile device and develop the application.

4.1 Mission Planner

In order to make the controller board functional it's needed to load the firmware. To do so, one program that can be installed is Mission Planner. It's a free, open source software available for windows with a large community of developers. This software was developed to be a GCS to helicopters, planes, multicopters and rovers. It can be used as a configuration utility or as a dynamic control supplement to the autonomous vehicle. Its main functionalities are: flight planning, its capability to install the firmware needed for some flight controllers, monitor the status of the vehicle during its operation, log the vehicle's activity, etc. To communicate with the controller board this software uses the same communication protocol as APM and Pixhawk, MAVLink.

After installing Mission Planner, the controller board should be connected to the computer with the help of a micro USB cable. The user must select the COM port from the drop-down box in the upper-right corner and make sure the right baud rate is selected, 115200 for USB or 57600 for Radio/Telemetry. After selecting the COM port the controller board can be connected by clicking in the connect button. To install the firmware the *initial setup* tab should be selected and next the user has to choose the firmware version to install, according to the type of vehicle desired (ArduRover for cars and boats, ArduPlane for fixed-wings and ArduCopter for multicopters).

To test if the firmware was installed successfully the connection to the controller board should be restarted by removing and inserting the USB cable. Once the connect button is clicked the parameters in the screen such as the altitude or the GPS status should update automatically according to the information transmitted by the board (Mission Planner Overview, n.d.).

In this dissertation Mission Planner will also be used to test the connection to the controller board once it has the Raspberry Pi connected and transmitting the parameters over the network by WI-Fi, 3G or 4G. To check if all the parameters are being correctly transmitted. To listen for packets coming from another wireless devices it's needed to select the UDP option in the upper right corner dropdown box, mentioned above, click in the connect button and enter the receiving port by default, port 14550.

4.2 MAVLink

Micro Air Vehicle Link (MAVLink) is a communication protocol used mostly in communications between GCS and UVs. It was first introduced in 2009 by Lorenz Meier under LGPL license and can be used to transmit the vehicle's altitude, speed, battery status, its GPS location, and other information. It was extensively tested on various controller boards like the PX4, Pixhawk and APM. A MAVLink message is just a stream of bytes encoded by the GCS to send to the controller board or vice-versa. The stream of bytes can be sent by USB serial or telemetry (both cannot be used at the same time, if the both are plugged in, USB is given preference over the Telemetry, which is ignored).

Content	STX	LEN	SEQ	SYS	COMP	MSG	PAYLOAD	CKA	CKB
Byte Index	0	1	2	3	4	5	6 to (n+6)	n+7	n+8

Table 4-1 – MAVLink message packet anatomy.

A MAVLink packet has the structure described in Table 4-1. The first six bytes are for the message header. The first byte, with index 0, indicated as *STX* in *Table 4-1*, is the packet start byte, which has a value of 0xFE and indicates the start of a new packet. *LEN* indicates the length of the payload and has a value of 0, up to 255. SEQ is the sequence number for each component, which allows to detect packet loss and has also a value of 0 to 255. *SYS* corresponds to the system identifier and indicates which system the message is coming from, this allows to identify different vehicles or GCS in the same network. *COMP* byte is the component identifier and allows to identify different sources from the same system, for example, two GCS applications running in the same device, each one of them has a unique component identifier. Both the system ID and the component ID have a range of values between 0 and 255. *MSG* is the last byte in the header and corresponds to the ID of the message sent, indicating the type of message, which allows to decode the payload correctly. The *PAYLOAD* is the actual data and depends on the type of message. Finally, the last two bytes are the checksum, which the software uses to check if the message is valid and not corrupted, which in this case, the system will discard the message. A MAVLink message has a minimum length of 8 bytes, usually in acknowledgement packets without payload and a maximum length of 263 for full payload messages (MAVLink Micro Air Vehicle Communication Protocol, n.d.).

To understand better how a MAVLink message is composed, the *HEARTBEAT* message type can be used as an example. This is one of the most important messages which the GCS keeps sending every one second to ensure that is in sync with the controller board and if a number of heartbeats fail the failsafe mechanism in the controller board is activated and the vehicle stops immediately or lands, in case of being a flying vehicle. Although the failsafe mechanism can be modified to have different behaviors. The *HEARTBEAT* message type has the ID=0 and is composed by six attributes, which are the following:

- Type: shows the firmware type installed, correspondent to the type of vehicle;
- Autopilot: the autopilot type, indicating the type of controller board (PX4, ArduPilot Mega, etc.);
- Base mode: this atribute indicates the current mode and only support the base modes, being equal in every controller boards running Ardupilot firmware;
- Custom Mode: indicates the modes that are specific to each controller board running Ardupilot;
- System Status: this attributes informs if the vehicle is calibrating, if is booting up, in standby or active, and others;
- MAVLink Version: indicating, as the name suggests, the MAVLink version.

There are many other type of messages, like the *GPS_RAW_INT*, with attributes like the longitude, latitude and altitude, the *VFR_HUD* indicating the speed, heading, altitude, and the climb rate of the UV and many other type of messages (MAVLINK Common Message Set, n.d.).

4.3 MAVProxy

MAVProxy is a command line GCS, it has the ability to forward the messages from the UV controller board over the network via UDP to other devices. It’s an open source tool and is programmed using Python. This software will installed in the Raspberry Pi connected to the controller board to be able to read its messages and forward them through the network to other devices. After installing MAVProxy in the Raspberry Pi, the controller board should be connected as shown in Figure 3-8. To test if the RPi and the APM are able to communicate correctly, they both should be powered and the following commands should be introduced in the RPi command line to open MAVProxy:

mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600

The '--master' indicates which port (serial, USB or network address/port) the controller board is connected, which should be changed according to the port being used. The "--baudrate" command is used only in serial links and specifies the baudrate limit.

Once the previous command is open the MAVProxy console should open. Now that the Raspberry Pi is successfully connected to the controller board and getting its information, it's needed to forward its messages over the network to a given device running a GCS application. To do so, it was created a private network for testing purposes. In this case this was done by using a router.

A computer running Mission Planner and the Raspberry Pi were both connected to the same network created. Next, the following commands were introduced in the MAVProxy console to forward the messages to the computer running Mission Planner:

output add 192.168.1.2:14550

The messages flow from the controller board to the computer running Mission Planner software is represented in Figure 4-1. Note that the IP written above is the private IP of the computer running Mission Planner, this should be changed according to the destination device private IP. The destination port is by default the 14550, used by Mission Planner to listen for MAVLink packets coming to the device.

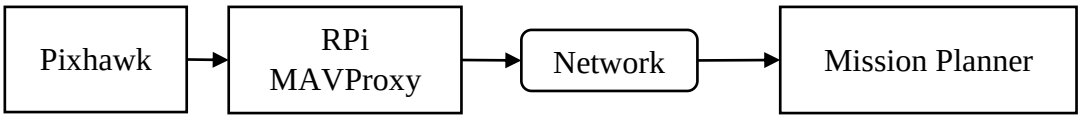


Figure 4-1 – Messages path from controller board to RPi.

4.4 SITL

Software In The Loop (SITL) is a software able to simulate the controller board (APM or Pixhawk) and was used in order to test the application being developed. This software is based on ArduPilot, allowing to run Plane, Copter or Rover without any hardware. SITL is especially useful since some malfunctions in the program while in the development phase can cause the vehicle to crash. This simulator is able to run either on Windows or Linux and require to install Cygwin which provides the tools and libraries that allows to

rebuild ArduPilot on Windows. Since SITL will emulate the controller board it will be required a way to communicate with it, to do so it will be installed MAVProxy alongside with SITL. After installing Cygwin and MAVProxy, SITL software is ready to launch, which could be done by opening Cygwin terminal and entered the following commands:

cd ~/ardupilot/ArduCopter

sim_vehicle.py –j4 –map

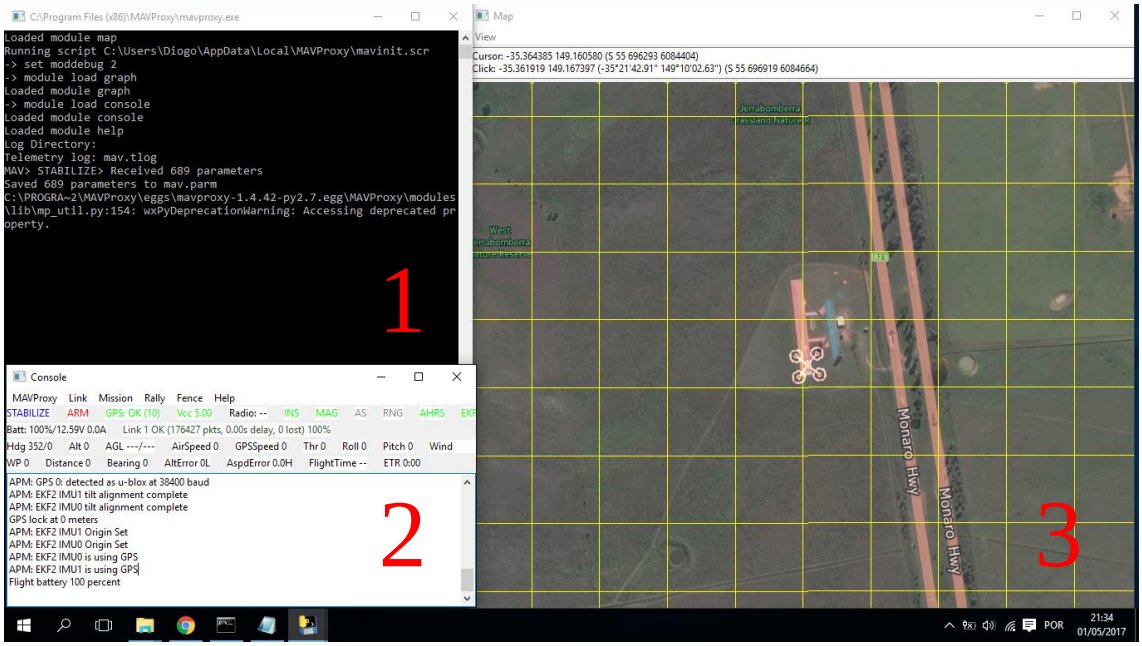


Figure 4-2 - Example of SITL and MAVProxy startup screens.

The commands shown above are applied to open SITL running with ArduCopter firmware, which will simulate a copter. To simulate a plane, "ArduCopter" should be replaced for "ArduPlane" or for "APMrover2" to simulate a rover. After the commands above are entered SITL and MAVProxy will open, displaying three windows as shown in Figure 4-2. First window is a command prompt in which the SITL commands will be entered (window number 1), a console displaying the current vehicle status messages (window number 2) and a map showing the vehicle's current position and direction (window number 3). The map can be used also to control and guide the vehicle movement by giving waypoints.

To connect SITL with the Mission Planner or other machine running a GCS software, the command described in the section 4.3 should be entered to forward the messages to the

correct devices. Due to the NAT protocol the destination machine should be in the same network as the machine running SITL. This was one of the challenges that must been overtaken in order to make the application work properly and communicate with the vehicle, being in different networks.

4.5 Network Address Translation

One of the goals of this dissertation is to be able to communicate with vehicles over long distances without line of sight. For this, the radio will not be sufficient. To overcome this, it was used Wi-Fi and 3G/4G communications. Wi-Fi was using the 2.4 and 5.15-5.35 GHz frequency bands which is an unlicensed spectrum in many countries. Each device connected to the internet has an IP address which is used for its identification in the network, which can be IPv4 or IPv6. Most of the times internet communications use a technique called Network Address Translation (NAT) which enables network administrators to use certain special addresses in their own network without having to request them from any service provider. The NAT creates a private network and acts like an interface, mediating the connection between that network and the public network, which makes the direct connection from one device to others in different networks more difficult to achieve. Another challenge is that the public IP of a device changes according to the network that it's in. Due to this fact the user (mobile application) doesn't know the IP of its vehicles. To overcome this problem the communication will be done using a server.

Each time a vehicle or an application is turned on, it will connect to an external server that will authenticate the device and register its IP, the server will also redirect the messages coming from the application to its vehicles and vice versa. The server implementation will be explained in more detail in the next chapter.

4.6 Video Streaming

The first method used for video streaming was the Wowza Streaming Engine, commonly used to stream live video and audio over IP networks to other devices. This service was developed in Java, uses Real Time Messaging Protocol (RTMP) and is compatible with most operating systems and browsers. To begin it was created a trial license account that has all the add-ons and the Wowza Media Server was installed in the server. The video coming from the camera was then redirected to the Wowza Media Server installed. Although, even after testing various scenarios and video configurations the delay was about 500ms which is a lot for real time control.

The next approach tested was video streaming via websockets. First a multimedia framework called *ffmpeg* needed to be installed in the Raspberry Pi in order to handle the video coming from the camera. This framework will encode the video to MPEG and send it to the server via HTTP. The server needs to have installed a JavaScript framework called *nodejs* to run a script that will distribute the MPEG stream via Websockets to the connected browsers that will decode it to reproduce the video (Szablewski, 2013).

To test the connection, HTML and JavaScript files were downloaded from *jsmpeg* project in GitHub. This files were open using the browser and they connect to the server via a websocket and display the image in a canvas. The latency using this method was much lower than using Wowza. The delay times will be explained further more in section 6.3.1.

Chapter 5
APPLICATIONS DEVELOPMENT

This chapter will explain how the server and the application were developed in order to achieve the purposed goals.

5.1 Server

The server was responsible for registering the users and their vehicles and exchange the messages between them. It was made in java and has two main components, the 'Dispatcher' and the 'ClientHandler' which will be explained latter. The server is located in France, has 2 GB of RAM, 2,4GHz of CPU speed and a 10GB SSD. The database used has a table named 'uvs' containing the information related to the vehicles and another table named 'users' containing the information related to the users. This two tables have a relation of many-to-many which require an intermediate table relating a user to an UV.

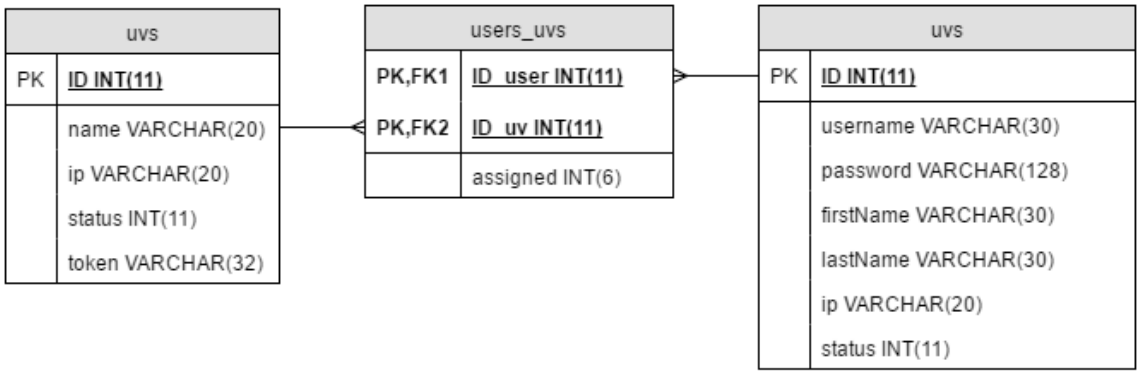


Figure 5-1- Database class diagram.

The server starts by opening a TCP socket in port 8009 to listen for any connections. When a client connects (coming from the application or the vehicle) the server starts a new 'ClientHandler' which is a Java object to refer users and UVs as clients and will handle with all the messages coming by TCP. There is a 'ClientHandler' instance for each vehicle and user connected. When the 'ClientHandler' corresponds to a UV, it will be responsible by authenticating the vehicle with its ID and token (a special key to each vehicle) and register its IP and owner. If the 'ClientHandler' corresponds to a user it will be responsible by authenticating the user with a username and password and each time a user (application) connects or asks for its own UV list, the 'ClientHandler' will return their connected vehicles with its ID's and IP addresses. When a user changes the selected vehicle to control, the 'ClientHandler' will receive that request and change the vehicle selected in the server, which will make the server redirect the messages coming from the user to the new vehicle selected. All the 'ClientHandler' messages, such as the login, the vehicle list and the change control messages need to be correctly delivered and must

ensure that they arrive at their destination. For this reason they are all sent over TCP, because this protocol is more reliable than other protocols such as UDP.

The server also opens an UDP socket in port 8010 and starts an object class called 'Dispatcher' to deal with all the MAVLink packages coming by UDP. The MAVLink messages come by UDP because they don't need to be retransmitted if an error occurs, which would cause more delay and they need to be delivered as soon as possible. The controller board is constantly sending messages. If one message gets lost it will quickly be replaced by other one. Each time a package comes over UDP the 'Dispatcher' starts by checking if the message came from an authenticated client. If the message came from an unauthenticated client the 'Dispatcher' discards the message. If it comes from an authenticated client it means that it has a 'ClientHandler' associated. In this case, it checks if the message was sent by the application or the vehicle. If the message corresponds to the application the 'Dispatcher' gets the 'ClientHandler' corresponding to the vehicle selected to be controlled by the user, as explained above, and send the message as it is to its corresponding IP address.

In case the message comes from a vehicle, the process is a little more complex. MAVProxy is constantly sending messages to their target devices, about 70 messages per second. If the owner has multiple vehicles connected and have a weak network connection, some packets can be lost. In a worst case scenario, if that packets correspond to the vehicle being controlled, this could easily compromise the safety of the vehicle control. To overtake this problem the following solution was implemented.

If the message comes from the vehicle being controlled (the selected vehicle by the user, as explained above) all the messages are redirected to the user. If the message comes from a vehicle that is not being controlled at the moment, only the MAVLink messages of the type GPS (containing the vehicle location), VFR_HUD (containing the velocity and pitch angle of the vehicle) and the HEARTBEAT message are sent to the corresponding user. This way, in case the user has multiple vehicles connected at the same time, the application is not overloaded with messages. Only the essential messages to see the vehicle's location and its state are needed in case it's not being controlled. All the messages corresponding to the vehicles need to be encapsulated in a package containing the IP of the vehicle who sent the message and then redirect the package to the corresponding user. This is necessary so that when the application receives the message

it can check the IP of the vehicle who sent it, allowing the application to distinguish between different vehicles in case it has multiple UVs connected.

As specified in Figure 5-1 there is also a status for the UV and the user. This variable is set to 1 if the device is connected and active and set to 0 if it's disconnected. In the intermediate table relating the UV and the user there is also a flag called "assigned" that is set to 1 if the user has selected its corresponding UV for controlling and is set to 0 if not.

5.2 Vehicle Application

A *jar* file is running in the RPi placed in the vehicle. This application is started each time the vehicle is turned on and is responsible by connecting the vehicle to the server. The application stores in a text file an individual ID and token that are different for each vehicle and they are used by the server to authenticate the vehicle to increase security. The applications listen and redirects to the server all the packages that are coming to the localhost port 14550 (127.0.0.1:14550) which is the IP and port that the MAVProxy sends the messages by default and vice-versa, redirecting the messages that are coming from the server to the MAVProxy (127.0.0.1:14550).

5.3 Mobile Application

The mobile application was developed in Android Studio. This operating system was chosen because it has many users on a global scale, it's free and is a native technology, having a better performance in the devices running Android. The minimum version targeted was *Jelly Bean 4.1* since it will run on 97,5% of android phones in the time being.

To be able to decode MAVLink messages it was used a library called MAVLink Java which decode the bytes with the help of *xml* files and generates Java code and vice-versa.

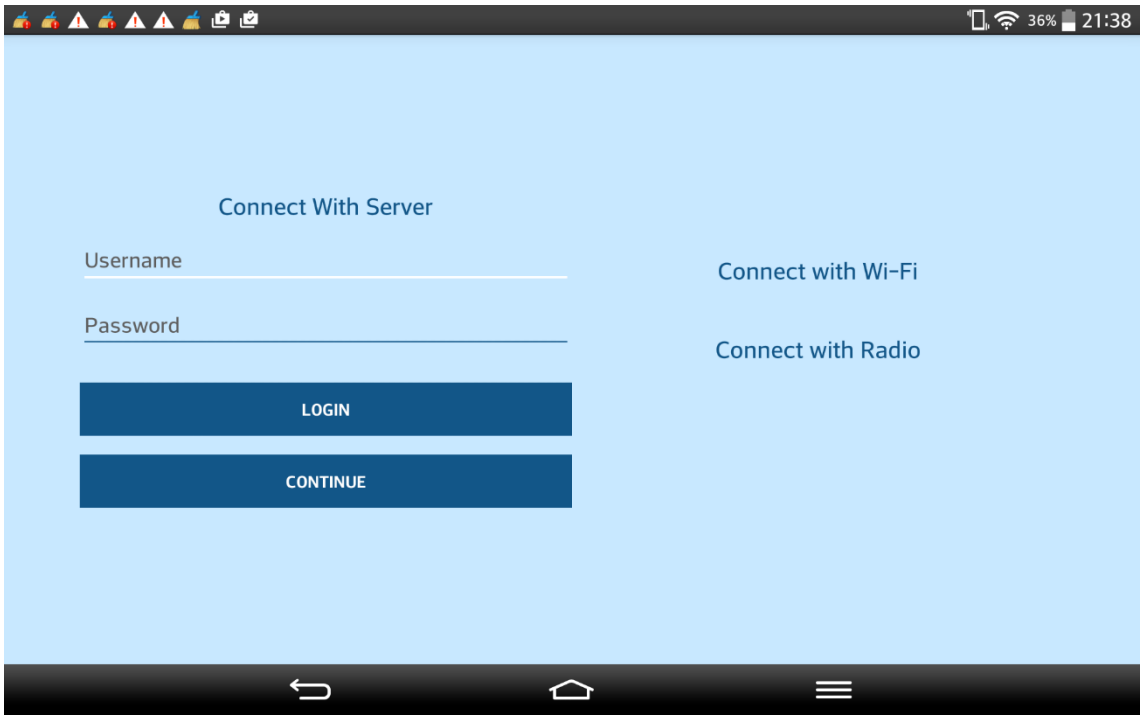


Figure 5-2 – Mobile control station login screen.

The mobile application called ‘Mobile Control Station’ starts in a login screen (Figure 5-2). In the login screen there are three options. The first one is called “Connect with Wi-Fi” and will make the application listen for MAVLink packages coming to port 14550 of the mobile device. This option can be very useful when testing the application with SITL since the user can add the device’s IP address to MAVProxy output list to redirect the messages to the application. Although, this require the user to know the IP address of the device which makes this option impossible to use in the real vehicles since they don’t know the mobile device’s IP address, reason by which the server was created. The second option is the “Connect with Radio” and allows the user to connect a radio dongle to the mobile device USB and to the controller board in the vehicle so they could exchange messages via radio. This was done with the aid of a Java library so the application was able to read the messages coming from the radio dongle. The radio is very useful to check the vehicle’s information, although the radio module has a high delay in the messages, about 400ms, and a short range which makes impossible the control of the vehicle in real time and BLOS. This two connect modes in the login screen were done to give the user more options to connect the UVs in different scenarios. Finally the third option is the “Connect with Server” which has a text area to introduce the username and password.

The application sends a login message to register the user in the server, sending its username and password. The password is then encoded in a MD5 hash and submitted to the server that will respond if the user is correctly authenticated. If the server responds with a successful message the user is considered logged in and redirected to the application's main screen which will receive the UV list sent by the server.

If there is any vehicles connected to the server, the server should start redirecting their messages to the UDP port opened by the application. As explained above, in the server section, the messages coming from the vehicles are encapsulated with the MAVLink message and the IP address of the vehicle who sent it. Once the application receives a message it will decode it, check the IP address of the vehicle who sent it and check which vehicle corresponds in the list returned by the server, to get its id and if it's an authenticated vehicle.

The application will then wait for a MAVLink message of type *HEARTBEAT* which has the information of the firmware version installed in the controller board, indicating the vehicle's type to add the UV to the side menu UV list. This is needed because the list has the icon corresponding to the type of vehicle. Then, the vehicle will be ready to be selected for the user to control.

The main screen has two tabs, the Controller tab (entry tab) and the missions tab, which will be explained latter.

Note that all the screens have popup messages to alert for errors that may occur during UV control.

5.3.1 Controller Screen

The controller tab redirects to a controller screen which is used to check the vehicle's information and manually control it.

Figure 5-3 – Controlling screen correspondent to the controller tab.

Label 1 in Figure 5-3 is the status button, which will be updated using the *HEARTBEAT* MAVLink message. This message has an attribute called "systems status", of type integer. Each mode of the vehicle has two "systems status" flags, one for when is armed and one for when it's disarmed. If the flag is lower than 150 the vehicle is disarmed, if the flag is higher the vehicle is armed. This button is updated according the vehicle status. Clicking the button will change the vehicle status by sending a MAVLink command message of type *MAV_CMD_COMPONENT_ARM_DISARM* with 0 to disarm and 1 to arm.

Label number 2 is the mode button and is used to change the mode of the selected vehicle. Each type of vehicle has its own modes. The Annex A – Vehicle's driving modes, described which modes are available in each vehicle type. The mode is returned by the *HEARTBEAT* message in its attribute named *custom_mode* and is used to update the button text. Clicking the button opens a popup list that allows the user to change the UV

mode. To do this, the application sends a MAVLink message of type *msg_set_mode* with the attribute *custom_mode* set to the corresponding value of the mode selected.

Label 3 indicates the UV battery level. This information is returned in the MAVLink message *SYS_STATUS* in its attribute *current_battery*. Although this only works properly if the controller board is powered via the *3DR Power Module* as explained in the section 3.2.4.

Presented by label 4 is the Round Trip Time (RTT) of the messages. This is the time the message takes to go from the application to the server and to the vehicle and vice-versa. There is a MAVLink ping message, although it does not work with the controller board used and neither with older controller boards. To overtake this problem the method implemented was to request for a parameter value to the controller board (the parameter request will be explained in more detail in the next sections), register the time when the request was sent and once the application receives the answer indicating the parameter value, it calculates the RTT which is given by the formula below:

$$EstimatedRTT = (\propto \times EstimatedRTT) + ((1-\propto) \times New\ RTT\ Sample) \qquad (5.1)$$

Where $\propto$ is the weighting factor, with values between 0 and 1. Choosing 1 will make the RTT immune to changes. Choosing 0 will make the RTT the value of the last round trip time calculated. In this case, the value displayed in the screen was the last RTT, with the $\propto$ value set to 0 so the user keeps updated with the last RTT.

The New RTT Sample is the difference between the time at which the message was received and the time that was sent. To being able to distinguish between requests, the parameter asked is different with each request.

The top bar at grey, label 5, indicates the height (in meters) of the vehicle, labeled as H(m). The UV's current height is always relative to its "home" altitude, where "home" is by default, the place where the UV was turn on. The second, labeled as 'VS', indicates the vertical speed in m/s, and the last one, labeled as 'HS', indicates the horizontal speed, in m/s, which is the climb speed. All this parameters are obtain via MAVLink message of type VFR_HUD.

On label 6 is placed the button which is used to activate the joysticks for manual control. Two joystick configuration was chosen because, after checking other applications in the market, this configuration is the most known and used in UV control. Using a design that is known, simplifies the use of the application because the users are already familiarized with the use of joysticks. The joysticks are common to every type of vehicle to simplify the use of the application. They are only activated if the vehicle is working in certain modes such as the Stabilize mode, Altitude Hold mode, Loiter or Manual mode because these are the only modes that the vehicle can be driven manually. The joysticks were implemented using a custom view that simulates a joystick. When in manual control, the application sends MAVLink messages of type *msg_rc_channels_override* were the attribute *chan1_raw* corresponds to the roll, which is the x axis of the right joystick by default, the second attribute is the *chan2_raw* which is the pitch value, the y axis of the right joystick. The *chan3_raw* and *chan4_raw* attributes of the message correspond to the thrust and yaw values, which are the y and x axis of the left joystick, correspondingly. Since all the values emulate the radio values their minimum should be set to 1000 and their maximum to 2000 which is the values in a real radio. One important aspect of the joysticks is that when the user clicks again in the button to deactivate the manual control the application releases the controls by setting the channel values to 0 during three seconds. This allows the user to use a radio controller at the same time of the application and if anything goes wrong the user can deactivate the manual control of the vehicle and will be able to control it with the radio controller. If the controls are not released by setting the channel values to 0 the user will never be able to control the vehicle in case there is a radio controller connected because as the type of message suggests (*msg_rc_channels_override*), the application overrides the radio controller channels. The control messages are sent with a frequency of 25Hz so if a message fails it will not compromise the control of the vehicle.

Label 7 corresponds to the Heads Up Display (HUD) which is overlaid to the video and shows the user the pitch, roll and the direction of the vehicle, as shown in Figure 5-4. The middle horizontal line is always aligned with the ground level, indicating the horizon.

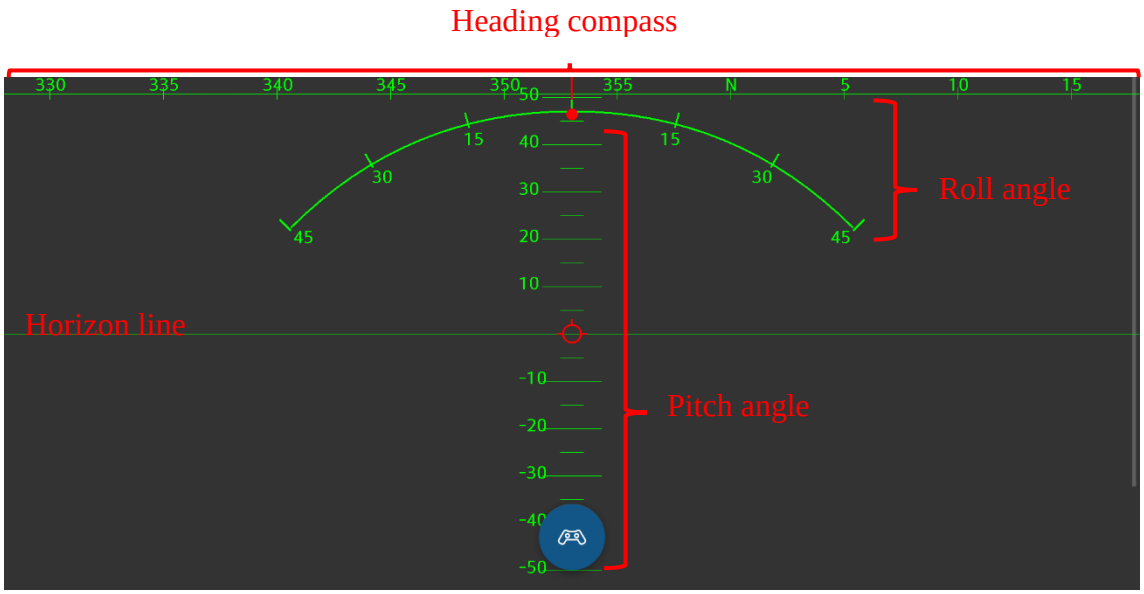


Figure 5-4 – Heads up display.

The video stream was loaded by using the HTML and JavaScript files described in section 4.6. It uses a Webview Java class that will receive the HTML file location and will connect to the server to load the live video transmition.

Label 8 is the icon which opens the side menu containing the vehicle list, the parameter list, the options and logout button.

5.3.2 Missions Screen

In the mission screen is possible to take off automatically to a certain altitude, give missions to the vehicle by sending waypoints, and land in the desired location.

Figure 5-5 - Missions screen correspondent to the missions tab.

There is also a HUD in the missions screen, displayed as label number 1 in Figure 5-5. With 'H(m)' indicating the altitude of the vehicle in meters, 'VS(m/s)' indicating the vertical velocity and 'HS(m/s)' indicating the horizontal speed.

Label 2 and 3 represents the arm/disarm button and the button to change the mode, similar to the ones used in control screen.

The request waypoints button (Label 4) is used to get the last waypoints sent to the controller board and show them on the screen. To request the waypoint list the application sends a MAVLink message of type *WAYPOINT_REQUEST_LIST* and the controller board responds with a message of type *WAYPOINT_COUNT* indicating the total number of waypoints saved. The application initiates a thread and asks for each waypoint separately by sending a *WAYPOINT_REQUEST(n)* where *n* is the number of the current waypoint being requested, starting in 0 and ending in the waypoint count returned previously. Each time the application asks for a waypoint it initiates a timeout timer and

if the waypoint is not returned within that time the application repeats the request for that waypoint. It does this process ten times before aborting the waypoint request list. The controller board responds with a MAVLink message of type *WAYPOINT* containing the waypoint number and location in latitude and longitude. Once the application receives that waypoint it asks for the next one until it reaches the waypoint count, sending a *WAYPOINT_ACK* message to the controller board indicating that all the waypoints where received with success. Each time the controller boards sends a waypoint it initiates a timeout timer, similar to the application, and if the application doesn't respond within that time asking for the next waypoint or with a ACK message, the controller board sends again the last waypoint asked. The waypoints requested are displayed in the application map by a blue color marker.

Label 5 is placed in the "Edit Waypoints" button which once activated allows the user to click in the map and add waypoints. The waypoints will then appear in a list where the user can click to edit the waypoint altitude or remove it. When the 'edit waypoints' button is activated there is also a 'clear' button that erases all the waypoints added and a 'send' button that will send the waypoints created to the controller board. The waypoints added will be displayed with a yellow marker.

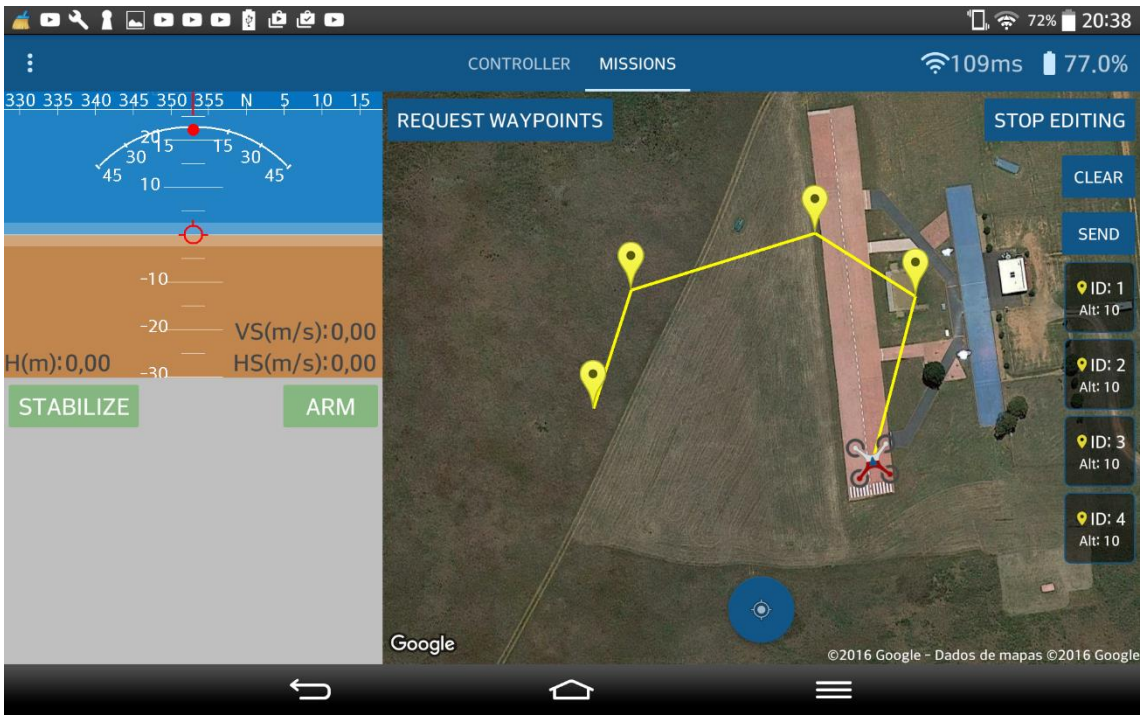


Figure 5-6 – Example of a screen with the waypoints being created.

To send the waypoints to the controller board the application starts by sending a MAVLink message of type *WAYPOINT_COUNT(n)* where *n* is the number of waypoints that will be sent. The controller board responds with a *WAYPOINT_REQUEST(n)* requesting the waypoint with the index *n*, starting in 0. The application will then respond with a WAYPOINT message correspondent to the waypoint asked, and once the controller board receives the answer it will ask for the next waypoint until it reaches the waypoint count sent by the application. Once the controller board receives the last waypoint it will respond with an ACK message to indicate that it received all the waypoints successfully. Every time a device sends a message (being the application or the controller board) it initiates a timeout timer and if it doesn't receive a response message within that time, it sends the last message again. If the timeout timer is reached 10 times in the application, it will abort the transition and sends an error message.

If all the waypoints are transmitted successfully the user can change to the "auto" mode, in the mode button (button label 3 in Figure 5-5), which will start the mission and the vehicle will move through the waypoints sent until it reaches the final waypoint where it will stay still. In case of being a flying vehicle, it will need to be in the air to start the mission. To do it the user can manually or automatically take-off.

To take-off automatically the user will need to arm the vehicle and change the mode to guided where it will appear a take-off button (label 6, Figure 5-5) with a place to select the desired height (that is 10m by default). Once the user clicks the button the vehicle will take-off automatically and stay in the altitude introduced. The take-off message is a MAVLink command message of type *MAV_CMD_NAV_TAKEOFF* which has an attribute called *param7* that is set to the introduced altitude in meters.

Making a long click in the map switches the vehicle mode to "guided" and sends a guided waypoint to the location clicked, sending the vehicle immediately to the desired location while maintaining its current altitude. The guided waypoint is displayed by a red marker in the application's map. Making a long click again in the map sends the vehicle immediately to the new location, even if the vehicle didn't reach the last guided waypoint or is in the middle of a mission. The guided waypoint is similar to the normal waypoint message but with the attribute called *current* that is set to value 2. In the normal waypoint message the *current* value is 0. The guided waypoint doesn't require to have the waypoint count sent first because it can only be sent one at a time.

The green waypoint in Figure 5-5 indicates the home position. This is by default the position where the vehicle was turn on. If the vehicle is not in the home position and is armed, switching to the RTL (Return To Launch) mode will guide the vehicle autonomously to the home position and in case of being a flying vehicle it will land autonomously. This is very useful when the vehicle is in open areas such as an open field but it's not advised to use in urban areas since the controller board cannot 'see' the environment and lead the vehicle against an obstacle in the way because the RTL takes the vehicle in a straight line to the home position. The user can also click in the waypoints to see its information.

5.3.3 Side Menu

In the side menu there is five options. The UV list, the Statistics screen, the parameter list, the settings and the logout button.

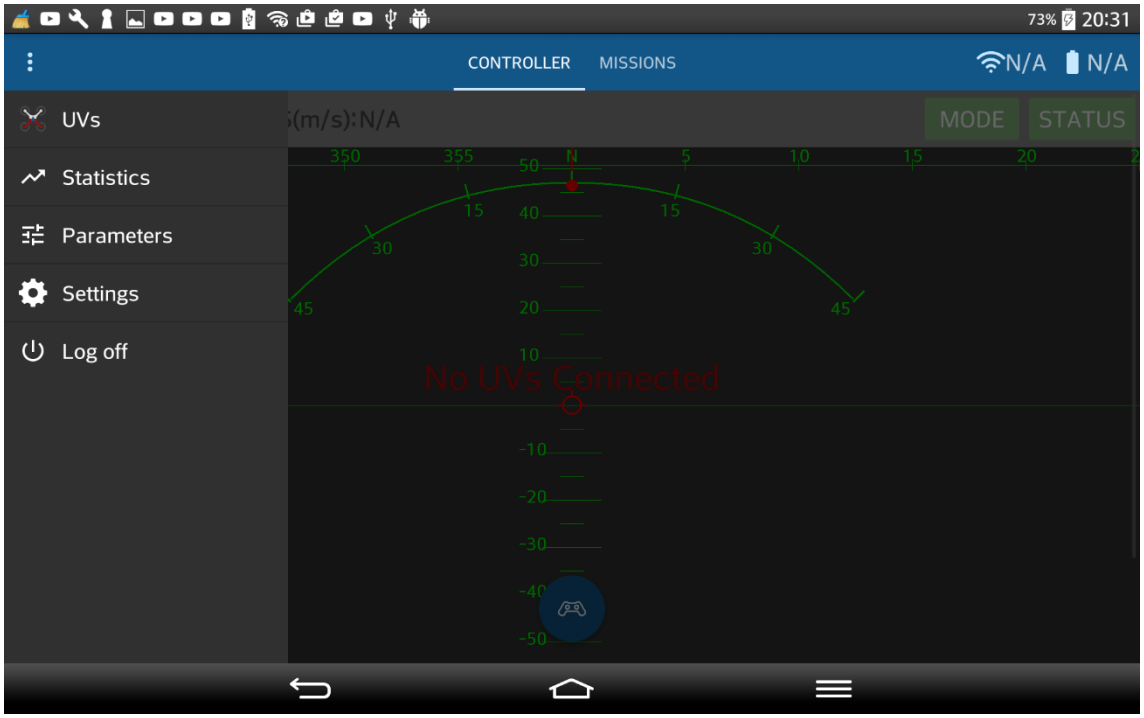


Figure 5-7 - Controlling Screen with side menu opened.

5.3.3.1 UVs List

The UV list (Figure 5-8), as explained above, is filled with the connected vehicles correspondent to the user that is logged in. If the applications stops receiving messages for 5 seconds the UV disconnects and it's erased from the UV list.

Figure 5-8 – UVs list popup.

The UV list is used to select the vehicle to control and sends a select message to the server that will change the assigned UV to control. The application displays an icon correspondent to the type of firmware of the controller board. It also displays the ID of the vehicle and the IP address and port so the user can distinguish between them.

5.3.3.2 Statistics Screen

The statistics option opens a statistics screen where there are displayed four graphics (*Figure 5-9*).

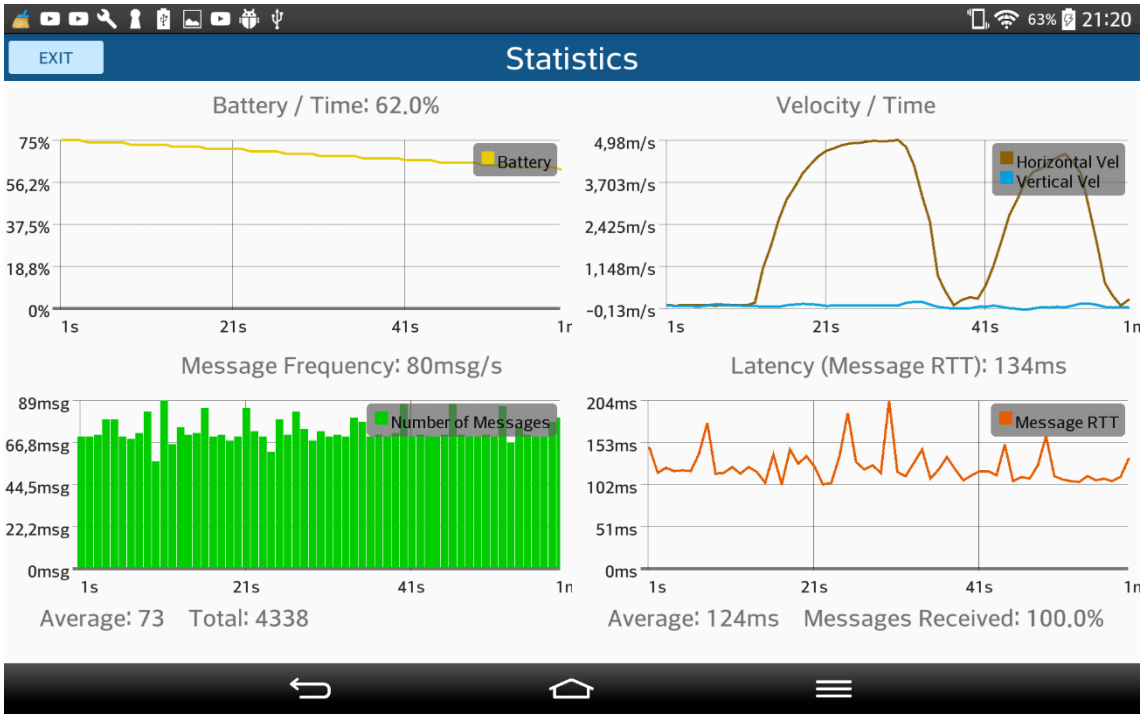


Figure 5-9 – Statistics screen.

The first one indicates the battery level over time. The second graph has two lines, a blue one for the vertical speed and a brown one for the horizontal speed, indicating the variation of speed over time. The third graph has the number of messages received over time and a label with the average number of messages per second and the total number of messages received since the application was initialized.

The last graph displays the RTT over time and the packet loss. The RTT is calculated with the formula explained in section 5.3.1. To get the packet loss, it's necessary to calculate the timeout time for a packet, which is calculated by the formula below:

$$Timeout = EstimatedRTT + 4 \times Deviation \quad (5.2)$$

$$Deviation = (1-\propto) \times Deviation + \propto \times |New\ RTT\ Sample - EstimatedRTT| \quad (5.3)$$

If a response message is not received within the timeout, is counted as a lost message. The percentage of messages received is calculated by the equation (5.4):

$$messages\ received = \frac{total\ ping\ messages\ sent - nº\ messages\ lost}{total\ ping\ messages\ sent} \times 100 \qquad (5.4)$$

5.3.3.3 Parameters Screen

As shown in Figure 5-10, the parameters screen show the list of parameters that is possible to edit.

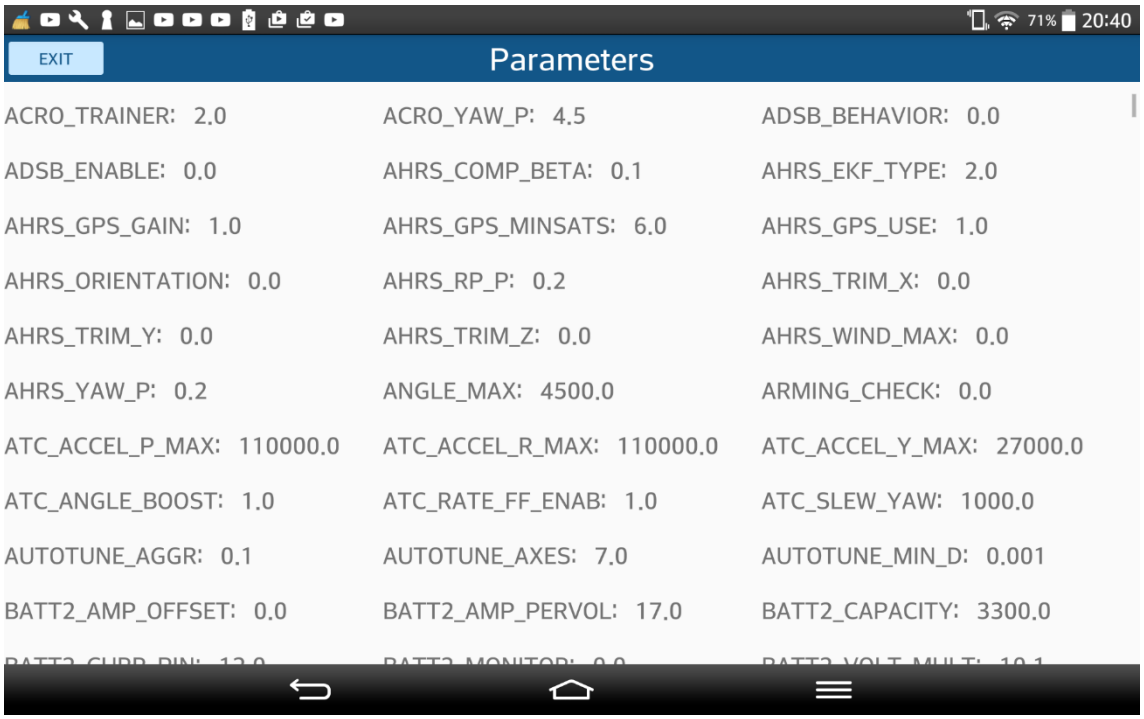


Figure 5-10 – Parameters screen.

To get the parameter list the application sends a *PARAM_REQUEST_LIST* and the controller board will send all the parameters individually. When the application receives the first parameter it will start a timeout timer and will add to a list all the parameters received. Each parameter contain an attribute named *param_count* indicating the total number of parameters and an *index* indicating the current parameter number. Once the application receives the last parameter (*index* = *param_count*) or the timeout timer is reached it will iterate the parameter list received and check for missing parameters. If there are missing parameters, due to packet loss, the application will ask for each one individually by sending a MAVLink *PARAM_REQUEST_READ* message that receives the index of the parameter being requested. The application does this until it has all the parameters. In case the timeout timer was reached 10 times, the application aborts the parameter request list operation and shows an error message.

Once the application has all the parameters it displays them in a screen where is possible to edit and save each parameter in the controller board. If the user clicks in the save button a MAVLink *PARAM_SET* message is sent to the controller board. This type of message contains the index of the parameter being edited and the value. After the controller board receives the *PARAM_SET* message it will return the parameter with the new value. If the application doesn't receive the parameter with the new values, it will resend the message until it receives the confirmation or the message was resent ten times, in this case the application will abort the parameter set operation and display an error message.

5.3.3.4 Options Screen

The options screen (Figure 5-11) has the options for each axis in the joystick. In this screen is possible to invert the controls, define the minimum and maximum of the axis and trim the middle value up or down like in a real radio controller.

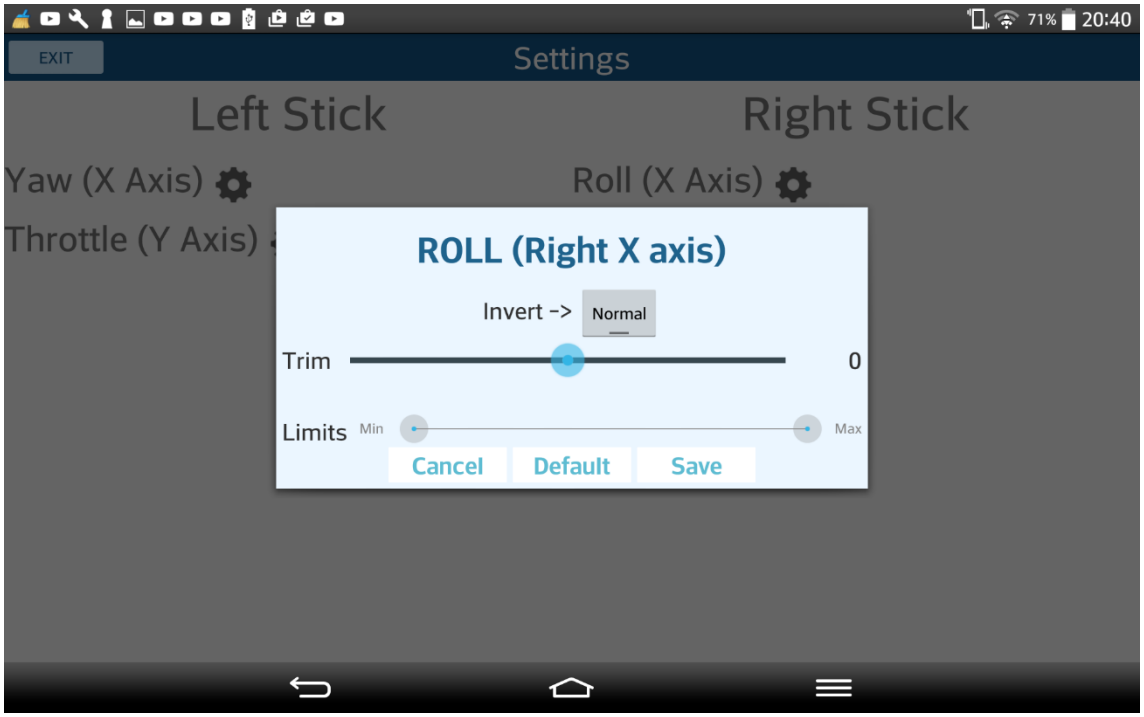


Figure 5-11 - Axis option window opened in the options screen.

5.3.3.5 Logout Option

Lastly the logout button sends a TCP message to disconnect the user. Once the server receives this message it will update the status of the user in the database and dissociate the vehicles connected.

Chapter 6
RESULTS

In this chapter will be shown all the results taken from the final tests to the application.

6.1 Tests Planning

To test the application and obtain the results in order to ensure that the application is functional and fulfills the purpose of this dissertation, a set of tests were realized in SITL and also with the real vehicles as can be seen in Annex B. The tests were divided in two phases, the first one to test the application main features, such as the monitoring and the controlling of the vehicle and the second to evaluate and analyze the delay times in the control messages and in the video feedback. For the control and the monitoring tests it were tested with the different types of firmware: rover, copter and plane. Although more exhaustive tests were done at the end of the application's development, the tests were always done throughout the application development to identify and resolve bugs and maximize the application's performance.

6.2 Monitoring Tests

The monitoring tests evaluate the ability of the application to monitor each vehicle in real time and show the user the information related to the vehicle selected, as well as sending commands so the vehicle can do autonomous tasks. To monitor a UV it was needed to log in the application. In the first phase it was tested the monitoring of each vehicle individually and the behavior of each mode. After entering the application, if a vehicle belonging to the user is already connected to the server, it will automatically be selected to monitor and will appear in the map, as described in Chapter 5.

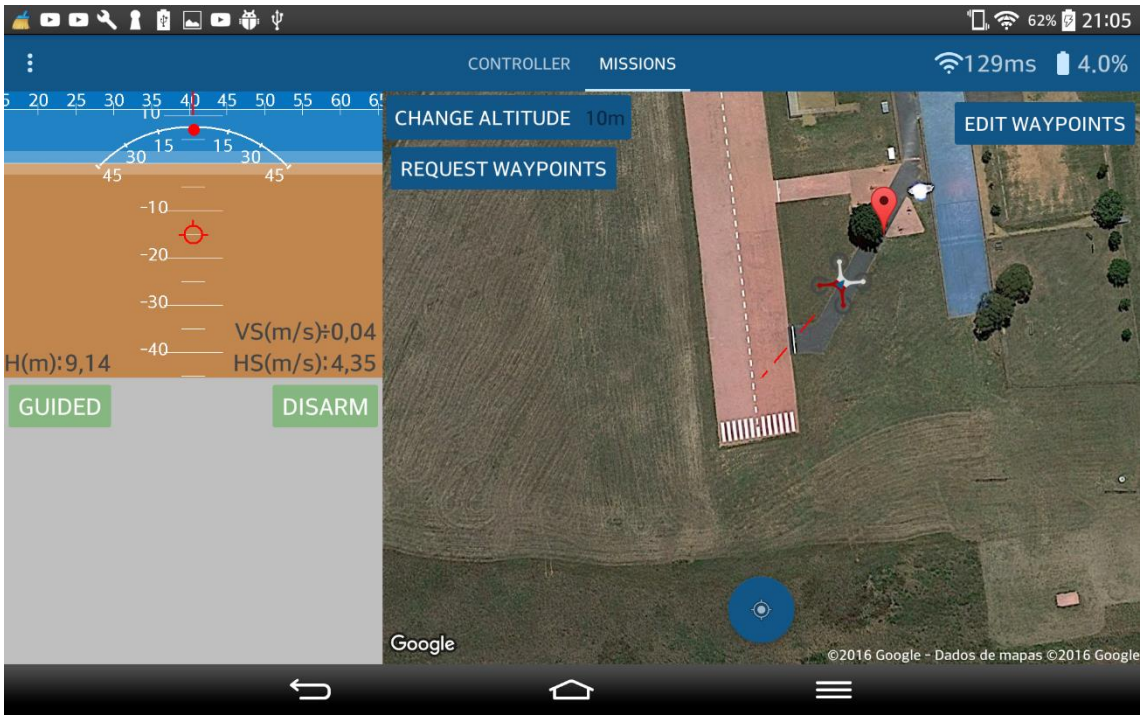


Figure 6-1 - Monitoring of a single drone

In Figure 6-1 a drone is connected to the application, the user is able to see the vehicle's remaining battery, the connection status, the current altitude, speed and location in real time, as well as the path traveled by the vehicle, which is marked by the red traces. The firmware tested in this example was ArduCopter, which is simulating a quadcopter. It were sent commands to the drone using the buttons on the screen. In Figure 6-1 the drone's mode is set to "guided".

6.2.1 Takeoff

One of the options tested while in guided mode is the automatic take-off, which it's explained in more detail on section 5.3.2. There were executed various tests using the take-off feature and changing the altitude. After sending the take-off command the drone will fly to the inserted altitude where it will remain flying while waiting for more commands. If the drone runs low on battery it will enable automatically the RTL (return to launch) mode. The take-off works best for copters and multi-copters because they are able to fly straight up and remain still in a desired position. The takeoff mode is not available for the ArduRover firmware.

6.2.2 Guided Mode

To test these feature with a quadcopter the UV must be flying. When a guided command is sent while the copter is landed, even if it's armed, the quadcopter will not respond. The speed at which the drone will fly could be defined in the parameters. As shown in Figure 6-1 the drone is flying to a guided waypoint, represented by the red marker. The quadcopter is flying at 4,35 m/s and with an altitude of 9,14 meters. The guided mode was also tested with ArduRover firmware, installed for the car and the boat. The rover only require to arm the vehicle before sending the guided waypoint.

6.2.3 Request Waypoints

As shown in Figure 6-2, after clicking in the "request waypoints" button, the application will retrieve the last waypoints saved in the controller board, which are displayed in the map as blue markers. These are the waypoints the vehicle will go though as soon as the user changes the mode to "Auto". The "request waypoints" feature was tested and proved to work properly in all the firmware's tested.

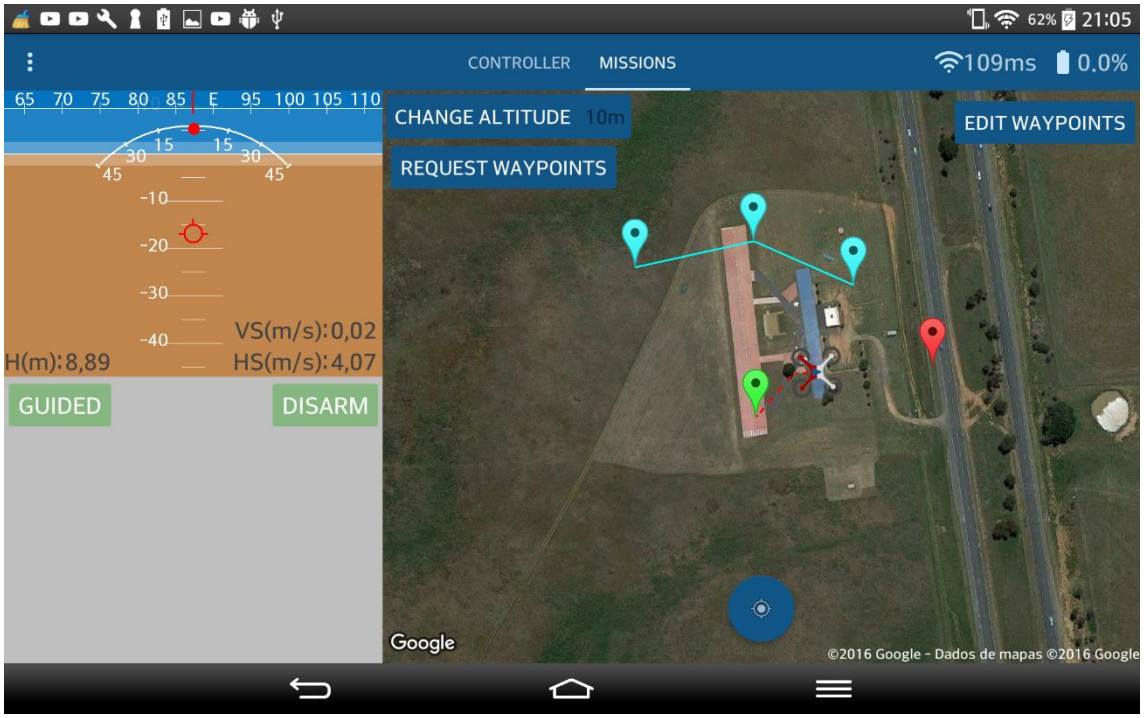


Figure 6-2 - Drone waypoints.

6.2.4 Auto Mode

To test the auto mode, which allows the user to send the vehicle in autonomous missions, some waypoints must be added. To do so, the button "edit waypoints" must be enabled. After clicking in the screen, a waypoint will be created in that location, which is indicated by a yellow marker as shown in Figure 6-3.

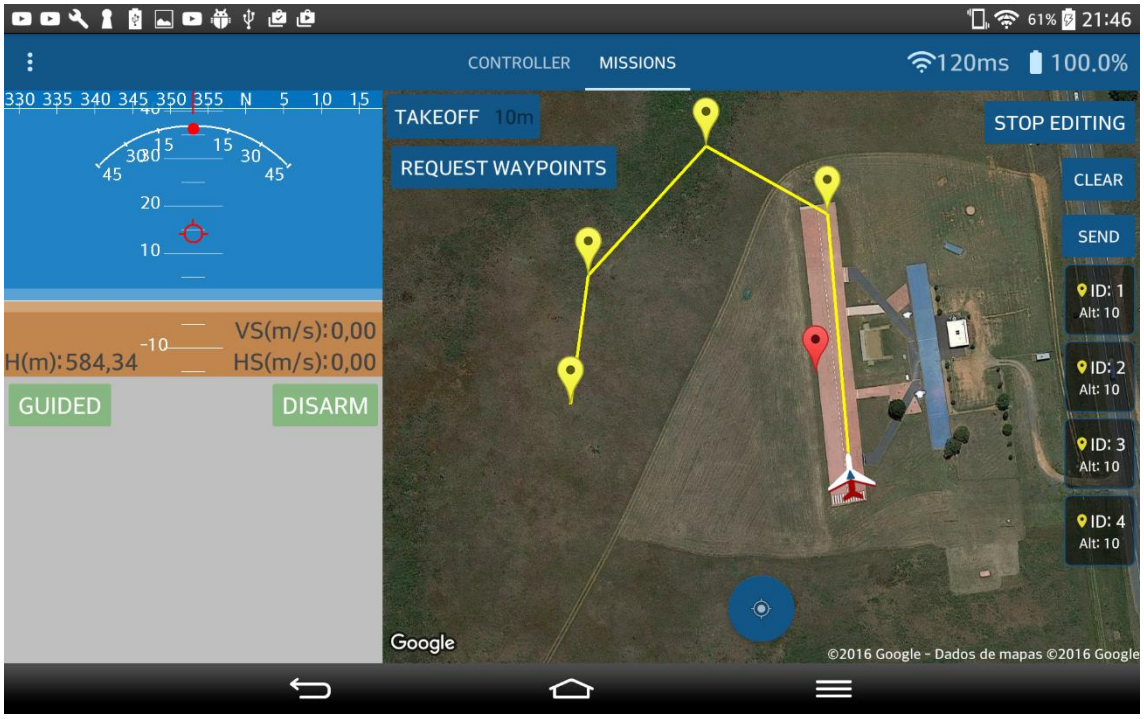


Figure 6-3- Editing waypoints.

The application will draw lines between the waypoints to show the user the path the drone will do once the user enables the auto mode. In Figure 6-3 four waypoints were added while the vehicle was landed, which can be seen by the four yellow markers, although, the waypoints can be added while the vehicle is flying or in an autonomous mission. The waypoints will be added to a list which will be displayed to the right of the screen, displaying the waypoint identifier and the altitude set for the waypoint. Clinking a waypoint in the list will give access to the waypoint options, displayed in a popup window, as shown in the image bellow (Figure 6-4).

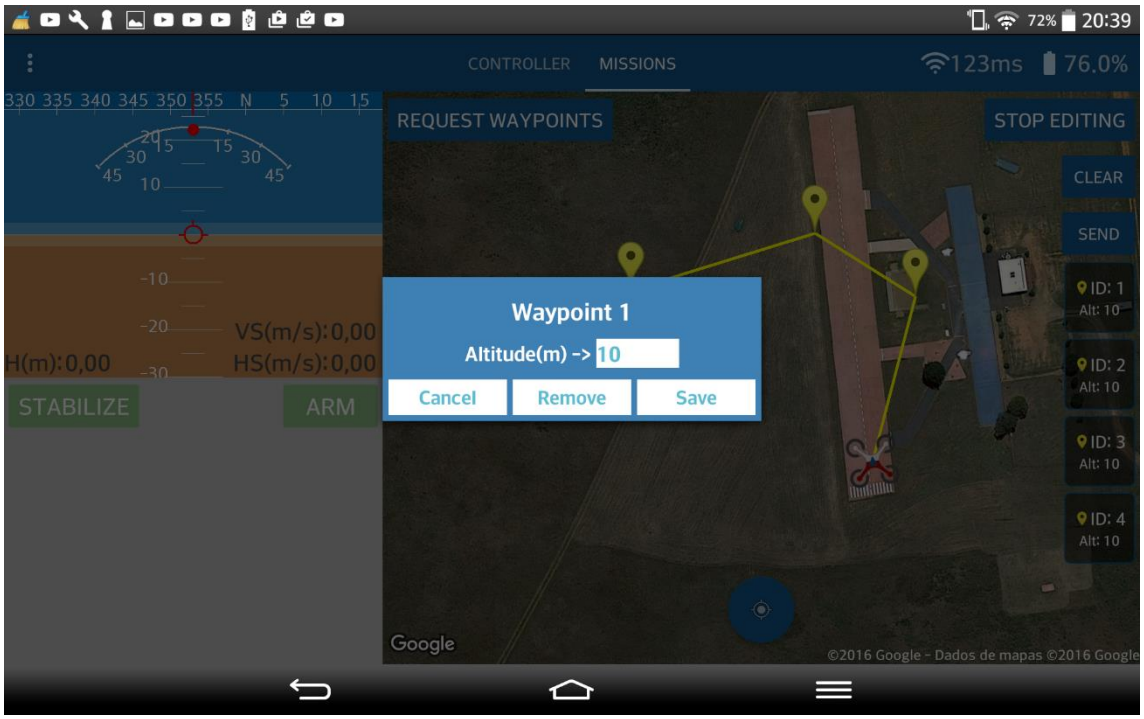


Figure 6-4 - Waypoints options.

In the waypoint options, the waypoint can be removed, by clicking in the ‘’Remove’’ button or the altitude of the waypoint can be changed, by clicking in the altitude text area and saving the configurations. The waypoint altitude will dictate the height at which the vehicle will pass. Several tests were performed by adding a list of waypoints to test the auto mode, which proved successful. Once the copter reaches a waypoint, it will face the next waypoint and move in a straight line. The minimum distance the vehicle has to reach the waypoint can be defined in the parameters list. For the tests this distance was set to 1 meter.

6.2.5 RTL

The result expected when testing the Return-To-Launch mode is when this mode is activated it will move the copter from its current position to the home point and land. When the RTL mode is enabled the copter will rise or descend to a certain altitude, which can be defined in the parameters, by the name of RTL_ALT, which by default is 15 meters, and move towards the home location. When the copter reaches the home location, it hovers for a certain time and then lands. The time the copter hovers the home positions before landing can be defined in the parameters by the name of RTL_LOIT_TIME.

The RTL works in a similar way for the plane. Although, with the rover firmware there is no need for the altitude.

6.2.6 Multi-Vehicle Monitoring Tests

After doing the monitoring tests with one vehicle, the tests were done with two and then three vehicles connected at the same time to the same device to test the reliability and behavior of monitoring various vehicles simultaneously. To realize this series of tests the SITL was connected to the server as a vehicle, using the java file, placed in the three computers running SITL, two of them running the ArduRover firmware and one running the ArduCopter firmware. The tests followed the same order as the tests done for one vehicle at a time.

After connecting the three vehicles to the server and connecting the application, the three vehicles will appear in a popup list so the user can select the vehicle to control as shown in the *Figure 6-5*. The current UV selected is the one running ArduCopter.

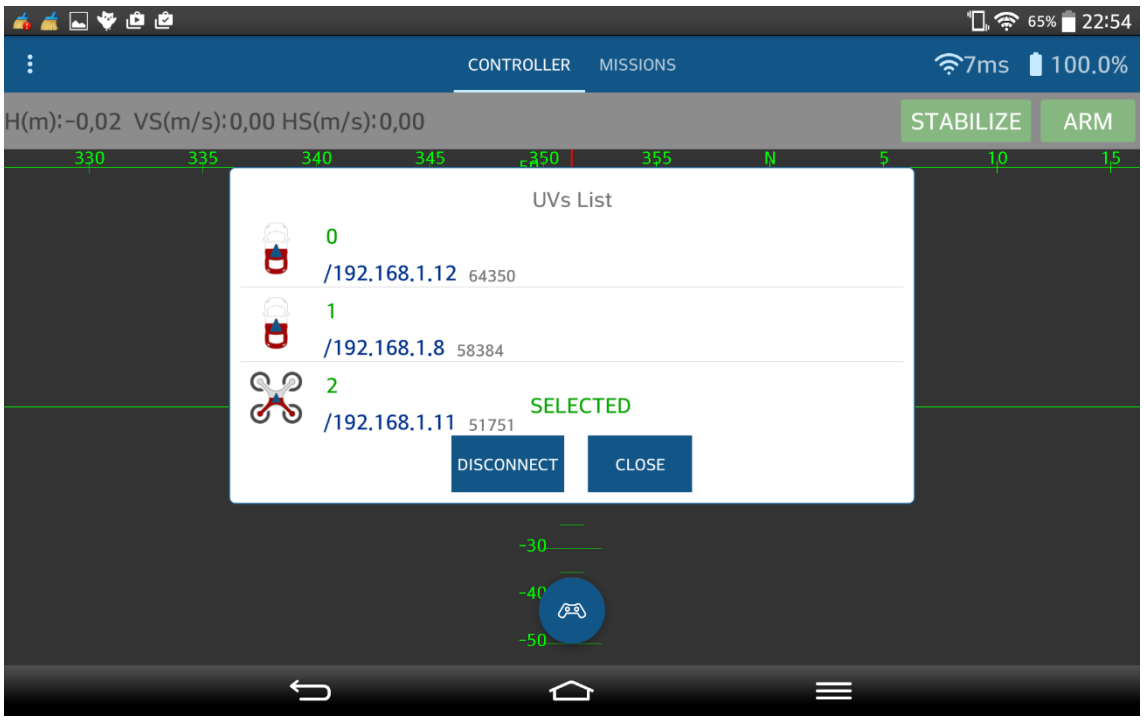


Figure 6-5 – UVs list with three vehicles connected.

After selecting a UV to control and entering the missions screen the user will be able to see and monitor all the vehicles connected, although it will only be able to control the

vehicle selected, meaning that all the comands and waypoitns added will only afect the current vehicle selected.

To test all the UVs connected, all the modes described in the monitoring tests of a single vehicle were realized for all the vehicles, one at a time, while the others were connected and being monitorized. Figure 6-6 shows the monitoring of these three vehicles connected.

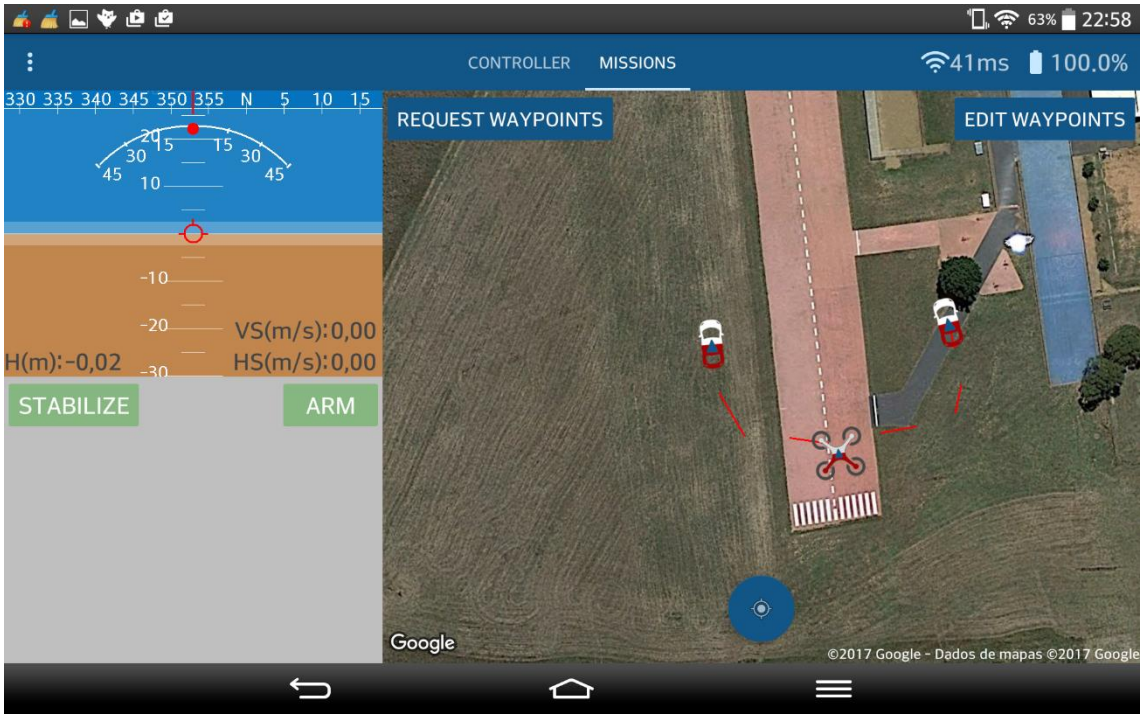


Figure 6-6 – Monitoring of three vehicles.

6.2.7 Evaluation

Analyzing the results and behaviors obtained from the tests realized and described in the monitoring tests, it was possible to conclude that the application allows the user to connect one or more vehicles simultaneously and monitor them, distinguishing successfully between the different types of vehicles and adapting the application to the different UVs and their different modes, allowing the user to select and control a single vehicle while monitoring the others. It was proven that the applications displays successfully the vehicles information, allowing the user to change the vehicle state (arm/disarm) and their operating mode.

6.3 Controlling Tests

The controlling tests were done in the controller screen, where the user has access to the video feedback and the joysticks to manually control the vehicle. They were realized both with a single vehicle connected as well with multiple vehicles connected simultaneously. Although, the behavior of the application in the controller screen, does not vary even with multiple vehicles, since the control screen will only present the information relative to the selected vehicle, as well as to control it.

Scenario	Test Case Description	Expected Result
Activate manual mode.	Enter application, change mode to "altitude hold" and click in the arm button.	Vehicle changes to altitude hold mode, pixhawk emits a 'beep' sound and changes led color to green or blue indicating that the vehicle is armed(check section 0)
ArduPilot Test (with hexacopter)		
Move copter up and down.	Test the throttle axis. Move left joystick up and down.	Copter ascends, gaining altitude when the left joystick is pushed up and descends when is down.
Move copter left and right.	Test the roll joystick axis. Move right joystick to the left and right.	Copter moves sideward, relatively to the side the joystick is pushed.
Move copter forward and backward.	Test the pitch joystick axis. Move right joystick forward and backward.	Copter moves forward and backward, leaning towards the side the joystick is pushed.
Turn copter.	Test the yaw joystick axis. Move left joystick left and right.	The copter rotates on its central axis, rotating clockwise when the left joystick is pushed to the right and anticlockwise when pushed to the left.
ArduRover Test (with car)		
Accelerate and Reverse.	Move left joystick up to accelerate and down to reverse.	When the joystick is above fifty percent (correspondent to the idle position) the car gains acceleration. When the joystick is below the middle position the car goes into reverse.
Turn the car's steering.	Move right joystick left and right according to the intended direction.	The direction of the car will point in the same direction as the joystick, the steering angle is also set by the joystick, the higher the angle of the joystick, the greater the steering angle.

Table 6-1 – Controlling test case table.

All of the tests described in Table 6-1 were done using the default joystick definitions. Although, it were done a set of tests using pre-defined settings as well as tests to the options menu, which are described in Table 6-2.

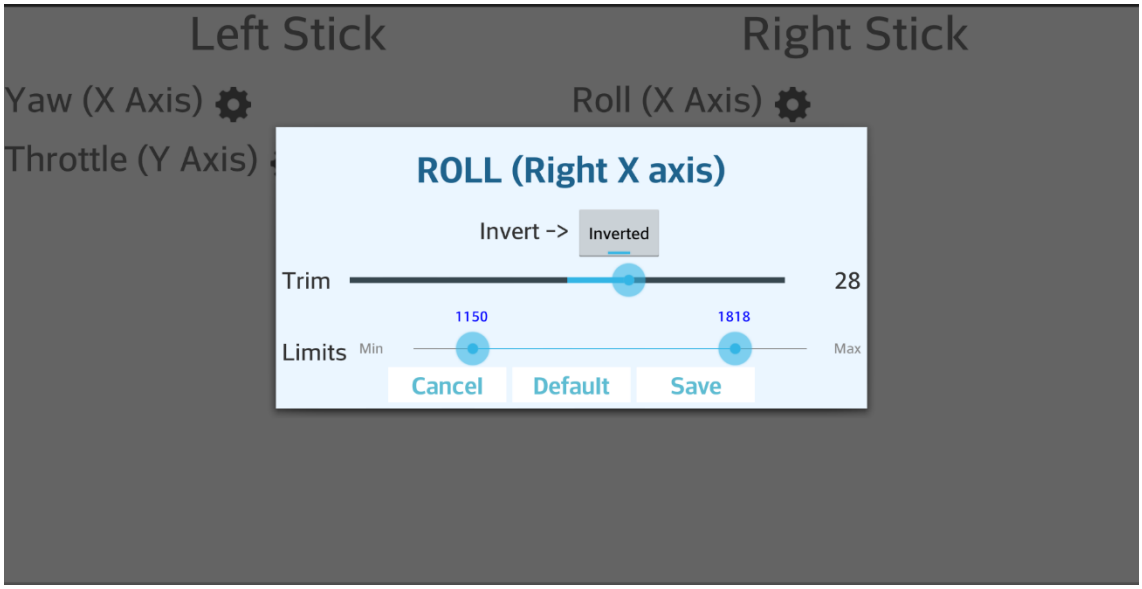


Figure 6-7 – Roll axis settings.

Scenario	Test Case Description	Expected Result
Enter joystick axis definitions.	Enter in the options screen by clicking in the options button in the side menu and click in the axis to edit.	Once the user enters the axis settings there will be a set of options so the user can define the settings.
Invert axis.	Click in the invert button.	If the axis is in its default state the button will be labeled as "Normal" and once the user clicks, the button will be reverted and will be labeled as "Revert".
Trim axis middle value.	Slide the blue button on the trim bar left or right.	If the button is slide to the left the middle button for the axis will decrease, being its default value 0 and its lower value -100. This is the value the axis will have once the button is released. If the button is slide to the right it will go from 0 to a maximum value of 100 (Figure 6-7).

Scenario	Test Case Description	Expected Result
Trim axis limit values.	Slide the *Min* button right and/or *Max* button left in the limit definitions bar.	If the *Min* button is slide right the joystick minimum value will grow, going from 1000 in its default value all the way to 2000 and the max button from 2000, being its default value, to 1000 (Figure 6-7). The *Max* button can never be lower than the *Min*.
Restore default values.	Click in the "Default" button.	All the settings will be restored to their default value. As shown in Figure 5-11.
Save settings	Click in "Save" button.	The user defined settings will be saved and the axis settings popup will close.

Table 6-2 - Controller options tests table.

Note that the user should have careful when editing the default values of the application since the controllers will have a different behavior, which could lead to a crash if not done properly. It's recommended to test in the simulator (SITL) first.

The control tests were done with the aid of the statistics screen in order to monitor the state of the network. This screen has an important role when the user wants to manually control the vehicle, as it allows him to monitor the amount of messages arriving at the application. Their periodicity, delay times and the network status. That is, if the network has connection problems or the messages RTT is too high (over 250ms), it is impossible drive the vehicle safely.

The figure below shows two print screens of the statistics screen running on different Wi-Fi networks. The first figure, Figure 6-8, shows a conjoined network, as can be seen by the failure of messages at certain points in time and the high RTT values. This can lead to the failure of manual control, so it is not advisable to control the vehicle manually in this type of networks. However, the monitoring can be used for missions or in the guided mode since once the waypoints are transmitted, the vehicle moves without the user interaction, and if it loses the connection and/or the UV has low battery, the vehicle is capable to land by itself, assuming the default settings. The second figure, *Figure 6-9*, shows a network with good conditions for manual control of the vehicle. The messages arrive at a steady pace and have a low RTT.

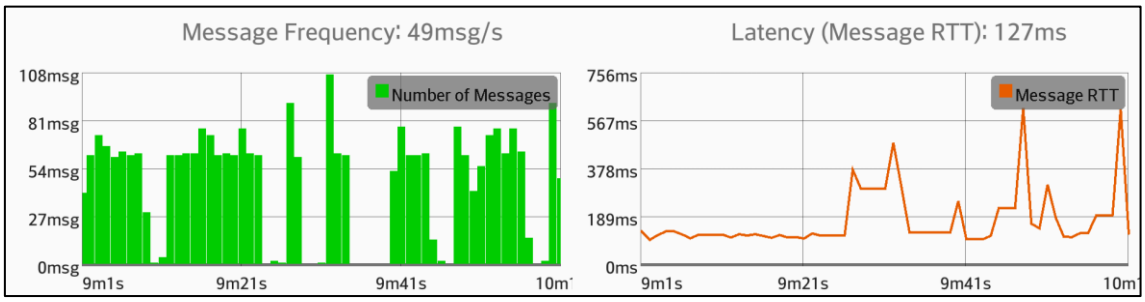


Figure 6-8 - Messages flow graphic on a conjoined network.

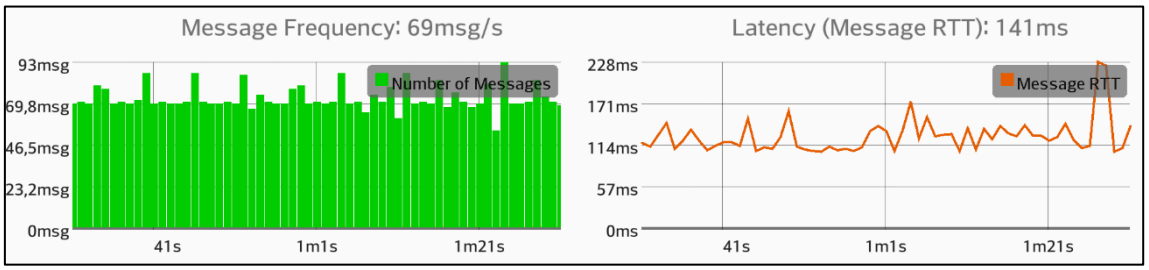


Figure 6-9 - Messages flow graphic on a good network.

The data of the statistics screen were collected over time to populate the following table showing the delay times in the vehicles control, with one and more vehicles connected to the application using the vehicle in a 4G network.

Video Feedback	1 Vehicle [ms]	2 Vehicle [ms]	3 Vehicle [ms]
No	127	134	142
Yes	172	188	210

Table 6-3 - Controlling messages RTT.

6.3.1 Video Streaming Tests

Several tests were also performed on the video stream that was transmitted from the vehicle to the server in order to help in its driving beyond line of sight. The tests were performed under three different conditions, on a 3G network, on a 4G network and then using Wi-Fi, to test all the control possibilities using wireless networks. The tests were realized using the Raspberry Pi with a camera connected and the SITL software. A technique was used to capture the delay times. This technique consists in filming a chronometer, placed in front of the RPi camera and taking a photo to catch the

chronometer and the tablet screen showing the video feedback, as can be seen in *Figure 6-10*.

Figure 6-10 - Test case of delay in the video stream.

Figure 6-10 shows the delay in the video stream using a 3G network. The camera is pointed at the chronometer that shows 1,38.06 seconds and the video stream in the application shows 1,37.68. Subtracting both values the value for the delay is around 0,38 seconds. Using this technique in the different scenarios it was possible to populate the following table.

Network Type	Resolution [pixels]	Framerate [fps]	Delay [ms]
3G	320x240	~24	360
4G	320x240	~24	330
Wi-Fi	320x240	~24	270

Table 6-4 - Table of results of delay times in the video feedback.

The video feedback case works differently than the MAVLink commands, for the video stream the vehicles send the video to the server and the application will only connect to the websocket containing the video feedback of the selected UV, so the fact of having

various vehicles connected will have little effect on the delay times in the video. Unlike MAVLink messages in which the server sends the information of all the vehicles that are connected to the server that belong to that user to the application.

6.3.2 Evaluation

By analyzing the results obtained from the controlling tests it's possible to conclude that it's safe and reliable to manually control the vehicle using wireless networks. Provided that the user ensures that there is a good internet connection and without failures, which can be easily seen in the statistics screen. The delay times for the vehicles control shown good results. In the control messages the delay is rarely noticeable, only the delay times over 200ms in RTT are perceivable. For the video stream the delay could be perceptible in certain conditions, especially in the 3G network. However, video settings can be changed to decrease delay times, such as reduce the resolution, frames per second or by changing the video color from RGB to black and white.

Chapter 7
CONCLUSIONS AND FUTURE WORK

The final chapter will describe the main conclusions obtained from the application development and ideas for future work.

7.1 Conclusions

Being the main goal of this dissertation to develop an application capable of controlling a 3D UV, through the development of the application described in this dissertation, it's possible to say that this objective has been accomplished. The application is also able to monitor not only a 3D UV, but several types of vehicles, with different architectures. Analyzing the results obtained through the tests realized to the application it is possible to withdraw several conclusions:

- The application provides the user an affordable and versatile way to control in real time various types of UV, as well as a 3D vehicle composed of several types of mobility systems;
- The communications protocol allows a reliable way to control one or more UV's using wireless networks, such as Wi-Fi, 3G or 4G, to send and receive the control messages and the video stream of each vehicle;
- Through the development of the server, it was possible to develop a secure and stable way to manage the users and their vehicles as well as to make the communication between them;
- The application, together with the controller board and the electronic systems of each vehicle, provide the user a simple way to monitor one or various UV's connected, check their status, information and change their settings.

7.2 Future Work

For future work related with this dissertation's scope, some suggestions are:

- Develop a 3D vehicle with a controller board and a firmware prepared exclusively for it, to avoid having to have a card for each type of vehicle;
- Improve and reduce the delay times in the video stream;
- Develop the application for other operating systems.

REFERENCES

3DR Power Module. (n.d.). Retrieved August 21, 2016, from Ardupilot: http://ardupilot.org/copter/docs/common-3dr-power-module.html#common-3dr-power-module

3DR UBlox GPS + Compass Module. (n.d.). Retrieved August 20, 2016, from Ardupilot.

A Guide to Understanding LiPo Batteries. (2016, January 6). Retrieved August 13, 2016, from Roger's Hobby Center: http://rogershobbycenter.com/lipoguide/

APM 2.5 and 2.6 Overview. (n.d.). Retrieved January 10, 2016, from Ardupilot: http://copter.ardupilot.com/wiki/common-apm25-and-26-overview/

APM 2.6 + Assembled. (n.d.). Retrieved January 10, 2016, from 3DR: https://store.3dr.com/products/apm-2-dot-6-plus-assembled-set-side-entry

Ata, W. G. (2014). *Intelligent Control of Tracked Vehicle Suspension.* PhD Thesis, University of Manchester, Manchester.

Balasubramanian, S. (n.d.). *MavLink Tutorial for Absolute Dummies*. Retrieved January 6, 2016, from Ardupilot: http://dev.ardupilot.com/wp-content/uploads/sites/6/2015/05/MAVLINK_FOR_DUMMIESPart1_v.1.1.pdf

Beginners' Guide Radio Control. (2016, August 15). Retrieved from Model Aircraft: http://adamone.rchomepage.com/guide1.htm

Brushless DC Motor, How it works ? (2016, August 13). Retrieved from Learn Engineering: http://www.learnengineering.org/2014/10/Brushless-DC-motor.html

Brushless motors - how they work and what the numbers mean. (2014, October 14). Retrieved August 13, 2016, from Drone Test: http://www.dronetrest.com/t/brushless-motors-how-they-work-and-what-the-numbers-mean/564

Büchi, R. (2012). Brushless Motors and Controllers.

Camera Module V2. (n.d.). Retrieved August 25, 2016, from Raspberry Pi: https://www.raspberrypi.org/products/camera-module-v2/

Communicating with Raspberry Pi via MAVLink. (n.d.). Retrieved January 12, 2016, from Ardupilot: http://dev.ardupilot.com/wiki/raspberry-pi-via-mavlink/

Connecting the Radio Receiver (APM2). (n.d.). Retrieved August 20, 2016, from Ardupilot: http://ardupilot.org/copter/docs/common-connecting-the-radio-receiver-apm2.html

DIY Drones Firmware builds. (n.d.). Retrieved Audust 11, 2016, from Ardupilot: http://firmware.ardupilot.org/

Do they both go round, mister? (n.d.). Retrieved January 11, 2016, from Model Boat Mayhen: http://www.modelboatmayhem.co.uk/Common/Electrics/Twin_motors.htm

Gage, D. W. (1995). UGV History 101: A Brief History of Unmanned Ground Vehicle (UGV). *Unmanned Systems Magazine*.

Hester, K. (Ed.). (2013, September 28). *Andropilot (arduleader) Wiki*. Retrieved January 10, 2016, from GitHub: https://github.com/geeksville/arduleader/wiki

HobbyKing Bixler 2 EPO 1500mm w/ Motor, Servos and Optional Flaps (ARF). (2016, January 11). Retrieved from HobbyKing: http://www.hobbyking.com/hobbyking/store/__24474__HobbyKing_Bixler_174_8482_2_EPO_1500mm_w_Motor_Servos_and_Optional_Flaps_ARF_.html

HobbyKing. (n.d.). *Turnigy 5000mAh 2S1P 7.4v 30C Hardcase Pack*. Retrieved August 14, 2016, from HobbyKing: http://www.hobbyking.com/hobbyking/store/__36054__Turnigy_5000mAh_2S1P_7_4v_30C_Hardcase_Pack_ROAR_APPROVED_EU_Warehouse_.html

Hornback, P. (1998). *The Wheel Versus Track Dilemma.*

Jenkins, D., & Vasigh, D. B. (2013). *The economic impact of unmanned aircraft systems integration in the United States.* Association for unmanned vehicle systems international.

LEDs (APM 2.x). (n.d.). Retrieved August 20, 2016, from Ardupilot: http://ardupilot.org/copter/docs/common-apm-board-leds.html

Loading Firmware onto Pixhawk/APM2.x/PX4. (n.d.). Retrieved August 11, 2016, from Ardupilot: http://ardupilot.org/copter/docs/common-loading-firmware-onto-pixhawk.html

MAVLINK Common Message Set. (n.d.). Retrieved November 03, 2016, from Pixhawk: https://pixhawk.ethz.ch/mavlink/

MAVLink Micro Air Vehicle Communication Protocol. (n.d.). Retrieved August 29, 2016, from QGroundControl: http://qgroundcontrol.org/mavlink/start

Mission Planner Overview. (n.d.). Retrieved August 12, 2016, from Ardupilot: http://ardupilot.org/planner/docs/mission-planner-overview.html

Mohammed, O. (2014). *A study of control systems for brushless DC motors.* University of Toledo, Toledo, Ohio.

Murilhas, L. C. (2015). *New system to drive future ground vehicles supported by heterogeneous wireless networks.* Master Thesis, ISCTE-IUL, Department of Information Science and Technology, Lisbon, Portugal.

NAZA-M V2. (n.d.). Retrieved August 12, 2016, from dji: http://www.dji.com/product/naza-m-v2

NCC Group Publication. (2016). *Creation of WiMap, the Wi-Fi Mapping Drone.*

Nehme, C. E. (2009). *Modeling human supervisory control in heterogeneous unmanned vehicle systems.* PhD Thesis, Massachusetts Institute Technology, Department of Aeronautics and Astronautics , Massachusetts.

Pi NoIR camera V2. (n.d.). Retrieved August 25, 2016, from Raspberry Pi: https://www.raspberrypi.org/products/pi-noir-camera-v2/

Pixhawk Autopilot. (n.d.). Retrieved August 12, 2016, from PX4 autopilot: https://pixhawk.org/modules/pixhawk

Pixhawk Autopilot. (n.d.). Retrieved December 20, 2016, from PX4 Autopilot: https://pixhawk.org/modules/pixhawk

Pixhawk Overview. (n.d.). Retrieved August 11, 2016, from Ardupilot: http://ardupilot.org/copter/docs/common-pixhawk-overview.html

Pixhawk Wiring Quick Start. (n.d.). Retrieved December 20, 2016, from Ardupilot: http://ardupilot.org/rover/docs/common-pixhawk-wiring-and-quick-start.html

Powering the APM2. (n.d.). Retrieved August 21, 2016, from Ardupilot: http://ardupilot.org/copter/docs/common-powering-the-apm2.html

Raspberry Pi 2 Model B. (n.d.). Retrieved August 22, 2016, from Raspberry Pi: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

Raspberry Pi Pinout Diagram | Circuit Notes. (n.d.). Retrieved August 22, 2016, from Jameco electronics: http://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html

Rönnberg, M. (2004). *Implementation of a Control System for a.* Master Thesis, Chalmers University of Technology, Department of Machine and Vehicle Systems, Gothenburg, Sweden.

Rover APM2.x Wiring and Quick Start. (n.d.). Retrieved August 20, 2016, from Ardupilot.

Saraiva, T. M. (2015). *Reliable Air-to-Ground Communication for Low Altitude Unnamed Aerial Vehicles.* Master's Thesis, ISCTE-IUL, Information Science and Technology, Lisbon, Portugal.

Scherer, M. (2015). *Development and Implementation of an Unmanned Aerial Vehicle with Stereoscopic Cameras Controlled via a Virtual Reality Head-Mounted Display Declaration of Authorship.* Master Thesis, Frankfurt University of Applied Sciences, Frankfurt, Germany.

Servo Anatomy. (n.d.). Retrieved August 14, 2016, from 2 Brothers Hobby: http://2bfly.com/knowledgebase/radio-systems/servos/servo-anatomy/

Setting up SITL on Windows. (n.d.). Retrieved September 23, 2016, from Ardupilot: http://plane.ardupilot.com/dev/docs/sitl-native-on-windows.html

Shamah, B. (1999). *Experimental Comparison of Skid Steering vs. Explicit Steering for Wheeled Mobile Robot.* Master Thesis, Carnegie Mellon University, Pittsburgh Pennsylvania.

Szablewski, D. (Ed.). (2013, September 11). *HTML5 live video streaming via websockets*. Retrieved December 6, 2016, from Photoslab: http://phoboslab.org/log/2013/09/html5-live-video-streaming-via-websockets

The effect of rudders, props and propwalk. (2012, June 29). Retrieved January 10, 2016, from RYA: http://www.rya.org.uk/cruising/handling-sail/Pages/TheeffectofrudderspropsandpropwalkonboathandIing.aspx

Turnigy D2836 / 8 1100KV Brushless Outrunner Motor. (n.d.). Retrieved 9 14, 2016, from HobbyKing: https://hobbyking.com/pt_pt/turnigy-d2836-8-1100kv-brushless-outrunner-motor.html

UAV Production Will Total $93 Billion. (2015, August 17). Retrieved from Teal Group Corporation: http://www.tealgroup.com/index.php/about-teal-group-corporation/press-releases/121-uav-production-will-total-93-billion

Unmanned Ground Vehicles (UGV) Market by Application (Defense, Commercial), Mode of Operation, Size, Mobility, Payload, and Region (North America, Europe, Asia-Pacific, Middle East, Rest of the World) - Global Forecasts to 2020. (2016, January). Retrieved January 10, 2016, from Markets and Markets: http://www.marketsandmarkets.com/Market-Reports/unmanned-ground-vehicles-market-72041795.html

Unmanned Surface Vehicles for Defense and Security: Global Markets and Technologies Outlook – 2013-2020. (2012, December). Retrieved January 10, 2016, from Market Info Group: http://marketinfogroup.com/unmanned-surface-vehicles-for-defense-and-security-markets-technologies/

Using a Standart USB Webcam. (n.d.). Retrieved August 25, 2016, from Raspberry Pi: https://www.raspberrypi.org/documentation/usage/webcams/

Vieira, P. (2014, November 3). *Basher Announces New 1/8 Scale BSR Rally*. Retrieved August 14, 2016, from Radio Control Car Action: http://www.rccaraction.com/blog/2014/11/03/basher-announces-new-18-scale-bsr-rally/

What are all those Knobs and Buttons on Your Drone's Transmitter Used for? (2016, January 5). (Brian, Producer) Retrieved November 13, 2016, from Fly Your Drones: http://flyyourdrones.com/drone-101/drone-controls/

What is a MultiCopter and How Does it Work. (n.d.). Retrieved from Ardupilot: http://ardupilot.org/copter/docs/what-is-a-multicopter-and-how-does-it-work.html

What is a Raspberry Pi. (n.d.). Retrieved January 2, 2016, from Raspberry Pi: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

ANNEXES

This page was intentionally left in blank

Annex A – Vehicle's driving modes

Mode Name	Mode Value
Stabilize	0
Acro	1
Alt Hold	2
Auto	3
Guided	4
Loiter	5
RTL	6
Circle	7
Pos Hold	8
Land	9
of_Loiter	10
Drift	11
Sport	13
Flip	14
Autotune	15
Pos Hold	16

Table A. 1 – ArduCopter firmware mode list.

Mode Name	Mode Value
Manual	0
Learning	2
Steering	3
Hold	4
Auto	10
RTL	11
Guided	15
Initialising	16

Table A. 2 – Mode list for APMrover2 firmware.

Mode Name	Mode Value
Manual	0
Circle	1
Stabilize	2
Training	3
Acro	4
FBW A	5
FBW B	6
Cruise	7
Autotune	8
Auto	10
RTL	11
Loiter	12
Land	14
Guided	15
Initialising	16
QStabilize	17
QHover	18
QLoiter	19
QLand	20

Table A. 3 – ArduPlane mode list.

Annex B – Tests in real vehicles

Figure B. 1 – Application being tested with a car, using APMrover2 firmware.

Figure B. 2 – Application tests in a quadcopter, using ArduCopter firmware.